

**UTILITY
PATENT APPLICATION
TRANSMITTAL**

Attorney Docket No.

750037.401C4

First Inventor or Application Identifier

D. David Nason

Title

**METHOD AND SYSTEM FOR CONTROLLING A
COMPLEMENTARY USER INTERFACE ON A DISPLAY SURFACE**

Express Mail Label No.

EL615486829US**APPLICATION ELEMENTS**

See MPEP chapter 600 concerning utility patent application contents.

ADDRESS TO:**Box Patent Application
Assistant Commissioner for Patents
Washington, D.C. 20231**1. ☐ General Authorization Form & Fee Transmittal
(Submit an original and a duplicate for fee processing)2. ☒ Specification [Total Pages] **67**
(preferred arrangement set forth below)

- Descriptive Title of the Invention
- Cross References to Related Applications
- Statement Regarding Fed sponsored R & D
- Reference to Microfiche Appendix
- Background of the Invention
- Brief Summary of the Invention
- Brief Description of the Drawings (if filed)
- Detailed Description
- Claim(s)
- Abstract of the Disclosure

3. ☒ Drawing(s) (35 USC 113) [Total Sheets] **34**4. Oath or Declaration [Total Pages] **1**

- a. ☐ Newly executed (original or copy)
- b. ☐ Copy from a prior application (37 CFR 1.63(d))
(for continuation/divisional with Box 17 completed)
 - i. ☐ **DELETION OF INVENTOR(S)**
Signed statement attached deleting
inventor(s) named in the prior application,
see 37 CFR 1.63(d)(2) and 1.33(b)

5. ☐ Incorporation By Reference (useable if box 4b is checked) The entire disclosure of the prior application, from which a copy of the oath or declaration is supplied under Box 4b, is considered to be part of the disclosure of the accompanying application and is hereby incorporated by reference therein.6. ☐ Microfiche Computer Program (Appendix)7. Nucleotide and Amino Acid Sequence Submission
(if applicable, all necessary)

- a. ☐ Computer-Readable Copy
- b. ☐ Paper Copy (identical to computer copy)
- c. ☐ Statement verifying identity of above copies

ACCOMPANYING APPLICATION PARTS8. ☐ Assignment Papers (cover sheet & document(s))9. ☐ 37 CFR 3.73(b) Statement (when there is an assignee) ☐ Power of Attorney10. ☐ English Translation Document (if applicable)11. ☐ Information Disclosure Statement (IDS)/PTO-1449 ☐ Copies of IDS Citations12. ☐ Preliminary Amendment13. ☒ Return Receipt Postcard14. ☐ Small Entity Statement(s) ☐ Statement filed in prior application, Status still proper and desired15. ☐ Certified Copy of Priority Document(s)
(if foreign priority is claimed)16. ☒ Other: Certificate of Express Mail
Appendices A-H

17. If a CONTINUING APPLICATION, check appropriate box and supply the requisite information below and in a preliminary amendment

☐ Continuation ☐ Divisional ☒ Continuation-In-Part (CIP) of prior Application No.: **09/666,032**

Prior application information: Examiner _____

Group / Art Unit _____

☒ Claims the benefit of Provisional Application Nos. **60/183,453, filed 2/18/00 and Attorney Docket No. 750037.401P4, filed 11/13/00****CORRESPONDENCE ADDRESS****00500**

PATENT TRADEMARK OFFICE

Ellen M. Bierman
Seed Intellectual Property Law Group PLLC
701 Fifth Avenue, Suite 6300
Seattle, Washington 98104-7092
Phone: (206) 622-4900
Fax: (206) 682-6031

Respectfully submitted,

TYPED or PRINTED NAME Ellen M. BiermanSIGNATURE Ellen M. BiermanREGISTRATION NO. 38,079Date Nov. 28, 2000

METHOD AND SYSTEM FOR CONTROLLING A COMPLEMENTARY USER INTERFACE ON A DISPLAY SURFACE

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is a continuation-in-part of U.S. Patent Application No. 09/666,032, filed on September 20, 2000, and claims the benefit of U.S. Provisional Application Nos. 60/183,453, filed on February 18, 2000 and _____, filed on November 13, 2000 (Attorney Docket No. 750037.401P4).

TECHNICAL FIELD

The present invention relates to a method and system for controlling the display of information on a display surface and, in particular, to computer software that displays one or more user interfaces that can coexist with a native user interface provided by the computer system.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram of a first embodiment of the present invention.

Fig. 2 is a block diagram of a second embodiment of the present invention.

Fig. 3 is a diagram of a standard display with an overscan user interface on all four borders of the display.

Fig. 4 is a block diagram of the basic components of a computer system video display environment that interacts with the methods and systems of the present invention.

Fig. 5 is a diagram of a cursor or pointer within the overscan user interface and the hotspot above it within the standard display.

Fig. 6 is a diagram of the usable border within the vertical overscan and the horizontal overscan surrounding the standard display.

Fig. 7 is an overview flow diagram showing the operation of a preferred embodiment of the present invention.

Fig. 8 is a flow diagram of the sub-steps in Identify Display step 102 of Fig. 7.

Fig. 9 is a flow diagram of the sub-steps of changing the display resolution step 114 of Fig. 7.

Fig. 10 is a flow diagram of the sub-steps in the Paint the Display step 120 of Fig.

5 7.

Fig. 11 is a flow diagram of the sub-steps of Enable Linear Addressing step 112 of Fig. 7.

Fig. 12 is a flow diagram of the sub-steps of the Process Message Loop of Fig. 7.

Fig. 13 is a flow diagram of the sub-steps of the Check Mouse and Keyboard Events step 184 in Fig. 12.

Fig. 14 is a flow diagram of the sub-steps of the Change Emulation Resolution step 115 in Fig. 7.

Fig. 15 is a diagram of a standard display of the prior art.

Fig. 16 is a diagram of a standard display with an overscan user interface in the bottom overscan area.

Fig. 17 is a diagram of a standard display including a desktop, an overscan user interface in the bottom overscan area and a context sensitive browser on the side.

Fig. 18 is a diagram of a standard display with an overscan user interface in the bottom and on the right overscan area.

Fig. 19 is a line drawing of a parallel GUI according to an example embodiment.

Fig. 20 is a simplified example of a menu tree.

Fig. 21 is a line drawing of a parallel GUI with an accessory container or cartridge.

Figs. 22-30 are example screen display illustrations of several complementary user interfaces coexisting with a native GUI.

Fig. 31 is an example block diagram of an implementation of the xSides™ architecture.

Fig. 32 is an example block diagram of an application using pixel mask technology in conjunction with an extended display area-enabled display driver.

Fig. 33 is an example screen display of application windows that are displayed using a universal trapping approach for modifying the display area and rendering outside of the native desktop.

Fig. 34 is an example block diagram of an example embodiment of the trapping technique for modifying the display area..

Fig. 35 is an example block diagram of the trapping architecture supporting multiple APIs for different windowing environments.

Fig. 36 is an example block diagram of the trapping architecture communication using kernel mode hooks.

Fig. 37 is an example block diagram of applications using techniques that intercept native graphics interface library calls.

SUMMARY OF THE INVENTION

Embodiments of the present invention provide computer-based methods and systems for displaying information on a display surface. When a native (resident) operating system is present, these embodiments display information in a manner that is complementary to the native operating system. The information displayed may be coexistent with a user interface provided by the native operating system. In addition, embodiments may be embedded into a native operating system and provide a primary interface to a display surface.

Embodiments also provide a technique for controlling allocation and content of display space among one or more user interfaces, operating systems or applications permitting an application or parallel graphical user interface (GUI) to operate outside the desktop, the area designated for display of the native operating system interface and it's associated applications. In an example embodiment, a computer operating under the control of any utility operating system such as Microsoft Windows™, Linux, Apple's Macintosh O/S or Unix may have the allocation of visible display controlled by techniques of the present invention. The operating system user interface (the native GUI) may be scaled and/or moved to a specific area of the display permitting a parallel (or complementary) GUI to operate in the open area. An example embodiment of the present invention may be as an application that operates under the primary or

utility operating system or it may be distributed as functionality that is combined with an operating system kernel (*e.g.*, distributed as a microkernel) to control the display and content in the parallel display.

Also, in some embodiments, a technique is provided for adding and using a parallel graphical user interface adjacent to the primary graphical display user interface, for example in the border beyond the standard screen display area. Conventional video systems, such as VGA, SVGA, XGA, SXGA and UXGA video systems, include a defined border surrounding the display area. The original purpose of this border was to allow adequate time for the horizontal and vertical retrace of the electron gun in a cathode ray tube display. However, with the advent of LCD displays and as retrace speeds have increased in modern monitors, it is now possible to present a user interface display in this border. The border which can be controlled as a user interface is a portion of what is known as the "overscan" area. Example embodiments include a method and system for presenting one or more additional or secondary user interfaces, for example, in the overscan area surrounding the native user interface display (the desktop).

When the electron gun in a CRT retraces to the left of the screen or the top of the screen, it requires a significant amount of time relative to the presentation of a scanned line of data. During the retrace, the electron gun is turned off ("blanked"). If the blanking time required for the retrace is equal to the amount of time available, there is no usable overscan. However, modern monitors have become much faster in their retrace speeds, leaving a significant amount of time when the electron gun need not be blanked, allowing a displayable border. In the prior art, although the border is usually "black" (the gun is turned off), it is well known how to specify that the border shall be given any one of six colors. Standard BIOS allows a specification of this color. The desired color is simply specified in one of the registers for the video controller. Typically no data for this color is stored in the buffer of video memory for the display. An example embodiment of the present invention establishes an additional video buffer for the border and allows this buffer to be written with display data like the regular display buffer. The additional video buffer is often present but unused in the graphics systems of most computers because video memory is usually implemented in sizes that are powers of 2, *e.g.*, "512K",

whereas standard desktop dimensions are not, “e.g., 640x480=300K”. The display area is thereby expanded, on one or more edges, to provide a visible area previously invisible. The pixels within this newly visible area of the display are made accessible to programs through an application programming interface (API) component of example embodiments of the present invention. A program incorporating a parallel graphical user interface may be displayed in the previously blanked area of the display, functionally increasing the accessible area of the display without hardware modification. In other cases the desktop may be increased or decreased to non-standard sizes to leave open display area for the parallel graphical user interface.

Example embodiments of the present invention include a method and system for displaying an image on a video display system in an area outside of the primary display area generated by the video display system by adjusting the video display area to include display memory outside of predefined video modes. Two dimensions define the standard display area, each specifying a number of pixels. Selecting a video “mode” specifies these dimensions. The method can be accomplished by adjusting parameters for the video display system to increase the number of pixels in at least one dimension of the display system. The number of pixels which is added is less than or equal to the difference between the number of pixels specified in the video mode and a maximum number of pixels which the video display system can effectively display. Any such difference is referred to here as an overscan area. Thus, the overscan area may be the difference between the current desktop video mode and the display capability of the display device or more specifically, any portion of video memory unused when the operating system is in a given screen dimension. Because most interface displays are created by writing a desired image to a buffer or memory for the video display, the method requires allocating additional video display memory for the added pixels. The image written to such memory is then displayed by the system alongside the original display area.

In other example embodiments, only the vertical dimension is increased and the parallel or complementary user interface is presented above or below the primary display area. Alternatively, the horizontal dimension may be increased and the parallel user interface displayed to the right or the left of the primary display area. Similarly, the parallel user interface may be displayed on any or all of the four sides of the primary display area.

In still other example embodiments, a parallel (or complementary) GUI is provided that includes access to existing search engines and browsers. In another embodiment, the parallel GUI includes a search engine and/or browser. A search engine and/or browser may be opened in either the overscan area or a space within or over the native operating system user interface.

In still other example embodiments, techniques are provided for adding and using a parallel graphical user interface adjacent to the primary graphical display user interface even if no overscan area is used. These techniques can be used to increase the overall real estate of the display area whereby the desktop is reduced, scaled, or moved to fit in a smaller or new portion of the total display area. A parallel user interface can then be displayed in the remaining portion of the total display area, or in a space within or over the desktop. In one embodiment, displaying and maintaining the parallel user interface is accomplished transparent to the user interface of the native operating system by intercepting calls to the video display driver. In some embodiments, techniques are provided for Windows™ environments and for Unix style environments. Other embodiments using similar techniques for other types of environments are also contemplated.

In yet another embodiment, a pixel mask technology is provided for supporting permitted applications to define, reserve, and use persistent display regions within the native desktop area of the display screen. These persistent display regions mask other output, thus preventing the output from the permitted applications to a persistent display region from being obscured by output from other (non-permitted) applications.

In yet another embodiment, a display-trap technology is provided to support a video card and driver independent mechanism for reducing the display area allocated to the desktop user interface, so that one or more parallel user interfaces can be displayed in the remaining area of the display screen.

In another embodiment, the methods and systems of the present invention are combined with voice and video streaming technologies, such as VoIP, IP streaming video, video encoding, video conferencing, and television programming and enabling technologies, such as EPG and HDTV support, to produce applications whose user interfaces communicate outside of the native desktop area. For example, calendars, calculators, video conferencing applications,

phones, etc. can be provided that are enabled to communicate with a user in one or more areas outside, or on top of, the desktop. In one embodiment, the user interfaces of these applications are persistent and operate independently of the native operating system, so that they remain executing, even when the operating system fails. In one embodiment, these applications are
 5 combined with a microkernel that is native operating system independent and can run on any computer system that the microkernel supports, including as an embedded application in a hardware device. In one embodiment, these techniques are used to create a webtop interface, which is independent of the desktop and the native operating system..

These and other features and advantages of embodiments of the present invention
 10 will become further apparent from the detailed description and accompanying figures and appendices that follow.

DESCRIPTION OF THE INVENTION

Embodiments of the present invention provide methods and systems for displaying information on a display surface in a manner that complements the display metaphor and technology provided by a native operating system. Using techniques of embodiments of the
 15 present invention, a complementary user interface is made operable within an existing system or is provided as a stand-alone environment. The complementary user interface may coexist as one or more secondary graphical user interfaces ("GUIs") with a primary user interface, such as conventional desktop GUI provided by the native operating system. The complementary user
 20 interface provided by such embodiments may be used, for example, to provide additional display screen real estate or to provide quick or continuous ("sticky") access to selected applications. The complementary user interface may provide access to a wide variety of capabilities, including, for example, continuous access to a user's favorite network locations on, for example, the Internet. For example, continuous access to applications such as a personal information
 25 manager, calendar, phone, video conferencing, television programming, etc. may be provided.

Referring now to Fig.'s 1 and 2, in a preferred embodiment, programming mechanisms and interfaces in a video display and control system such as computer system 7 or settop box 8 provide one or more parallel GUIs such as space 2C and/or space 4 in a display area

such as display area 1 or display area 9 by providing access and visibility to a portion of the display otherwise ignored and/or inaccessible (an "overscan area"). Display areas such as display area 1 or display area 9 may be created on any type of analog or digital display hardware including but not limited to CRT, TFT, LCD and flat panel displays.

5 Alternate display content controller 6 interacts with the computer utility operating system 5B and hardware drivers 5C to control allocation of display space 1 and create and control one or more parallel graphical user interfaces such as context sensitive network browser (CSNB) 2 and internet pages 2A and 2B adjacent to the operating system desktop 3. Alternate display content controller 6 may be incorporated in either hardware or software. As software, an
 10 alternate display content controller may be an application running on the computer operating system, or may include an operating system kernel of varying complexity ranging from dependent on the native operating system for hardware system services to a parallel system independent of the native operating system and capable of supporting dedicated applications. Applications enabled with the alternate display content controller also may be embedded in
 15 various devices. The alternate display content controller may also include content and operating software such as JAVA delivered over the Internet I, or over any other network.

The alternate display content controller may also be included in a television decoder/settop box such as box 8 to permit two or more parallel graphical user interfaces such as pages 9A and 9B to be displayed simultaneously. Methods and systems of the present invention
 20 may be compatible with conventional television formats such as NTSC, PAL, PAL-C, SECAM and MESECAM. In this configuration content and software may be delivered over any conventional delivery medium 10 including but not limited to over the air broadcast signals 10A, cable 10C, optical fiber, and satellite 10B.

Fig.'s 1 and 2 will be referenced in more detail below.

25 Fig. 15 shows an example of a standard prior art display desktop generated by a Microsoft Windows 95™ operating system. Within the desktop 31 are the taskbar 32 and desktop icons 33.

In one embodiment of the present invention, a complementary graphical user interface image is painted onto one or more of the sides of the overscan area as shown in Fig. 3.

Fig. 3 is a depiction of a Super VGA (SVGA) display with the addition of a graphical bar user interface displayed in the overscan area. The overscan user interface bar 30 is defined to reside outside the borders of the "desktop" display area 31. In Fig. 16, the display is modified to include a graphical user interface 30 in a bar 20-pixels high below the bottom edge. In Fig. 3, the display is modified to include a graphical user interface in four bars each 25-pixels high/wide outside each of the four display edges: a bottom bar 30, a left side bar 34, a right side bar 36, and a top bar 38. The complementary interface may include, and is not limited to, buttons, menus, application output controls (such as a "ticker window"), animations, and user input controls (such as edit boxes). Because the complementary interface is not obscured by other applications running within the standard desktop, the complementary interface may be constantly visible or it may toggle between visible and invisible states based upon any of a number of programming parameters (including, but not limited to, the state of the active window, the state of a toggle button, a network message, user preference, *etc.*).

Also, once the overscan area is allocated or other methods are employed to increase the overall real estate of the display area (or even if the display area remains unchanged in size), the native desktop may be reduced or moved to fit in a smaller or new portion of the total display area, leaving any side or other region open for displaying the complementary user interface. Figures 22-30 illustrate several possible configurations and allocations of the display area to include one or more complementary user interfaces. These figures illustrate that the complementary user interfaces may have heterogeneous styles and sizes and may reside on one or more areas of the overscan area as well as within (overlying) the native GUI (see, for example, menus 2301 in Fig. 23). In addition, the desktop may be moved or reduced, as shown in Fig. 25 and 30, and used in conjunction with complementary user interfaces that reside outside of or within the modified desktop. Figure 25 also demonstrates a complementary GUI whose contents can be dynamically driven by connecting to a network, such as the Internet. One skilled in the art will appreciate that any combination of these features is possible in practicing embodiments of the present invention and that additional features may be added within the scope of the present invention.

1. Video Display System Environment

Fig. 4 is a block diagram of the basic components of a computer system video display environment that interacts with the methods and systems of the present invention. Within the software component S are the native operating system 63 and one or more applications such as application 61. Within the protected modes of modern systems, applications 61 do not have direct access to the Video or Graphics Drivers 64 or hardware components such as the video card 66 which contains the video chipset 66A, 66B and 66C. Abstraction layers such as Application Programming Interface (API) 60, and/or DirectX API 62, provide limited access, often through the operating system 63. One such example API is Microsoft's GDI, which provides graphical display capabilities to applications running in Microsoft Windows environments.

Embodiments of the present invention provide a technique for painting and accessing an area of the computer display not accessible, or used, in the native desktop graphics modes. In the Microsoft Windows environments (including Microsoft Window 95 and derivatives, and Microsoft Windows NT 4.0 and derivatives) and other contemporary operating environments, the primary display area desktop is usually assigned by the operating system to be one of a set of pre-determined video "modes" such as those laid out in Tables 1 and 2 below, each of which is predefined at a specific pixel resolution. Thus, the accessible area of the computer display may not be modified except by selecting another of the available predefined modes.

TABLE 1: ROM BIOS VIDEO MODES

Mode Number	Resolution	Mode Colors	Buffer Type	Segment
00H	42x25 chars (320x350 pixels)	16	Alpha	B800
00H	42x25 chars (320x350 pixels)	16	Alpha	B800
00H	42x25 chars (320x400 pixels)	16	Alpha	B800
00H	42x25 chars (320x400 pixels)	16	Alpha	B800
01H	42x25 chars (320x200 pixels)	16	Alpha	B800
01H	42x25 chars (320x350 pixels)	16	Alpha	B800
01H	42x25 chars (320x400 pixels)	16	Alpha	B800
01H	42x25 chars (320x400 pixels)	16	Alpha	B800

02H	80x25 chars (640x200 pixels)	16	Alpha	B800
02H	80x25 chars (640x350 pixels)	16	Alpha	B800
02H	80x25 chars (640x400 pixels)	16	Alpha	B800
02H	80x25 chars (640x400 pixels)	16	Alpha	B800
03H	80x25 chars (640x200 pixels)	16	Alpha	B800
03H	80x25 chars (640x350 pixels)	16	Alpha	B800
03H	80x25 chars (640x400 pixels)	16	Alpha	B800
03H	80x25 chars (720x400 pixels)	16	Alpha	B800
04H	320x200 pixels	4	Graphics	B800
05H	320x200 pixels	4	Graphics	B800
06H	840X200 pixels	2	Graphics	B800
07H	80x25 chars (720x350 pixels)	2	Alpha	B000
07H	80x25 chars (720x400 pixels)	2	Alpha	B000
0DH	320x200 pixels	16	Graphics	A000
0EH	640x200 pixels	16	Graphics	A000
0FH	640x350 pixels	4	Graphics	A000
10H	640x350 pixels	4	Graphics	A000
10H	640x350 pixels	16	Graphics	A000
11H	640x480 pixels	2	Graphics	A000
12H	640x480 pixels	16	Graphics	A000
13H	320x200 pixels	256	Graphics	A000

TABLE 2: SVGA VIDEO MODES DEFINED IN THE VESA BIOS EXTENSION

Mode Number	Resolution	Mode Colors	Buffer Type
100H	640x480 pixels	256	Graphics
101H	640x480 pixels	256	Graphics
102H	800x600 pixels	16	Graphics
103H	800x600 pixels	256	Graphics
104H	1024x768 pixels	16	Graphics
105H	1024x768 pixels	256	Graphics

106H	1280x1024 pixels	16	Graphics
107H	1280x1024 pixels	256	Graphics
108H	80x60 chars	16	Alpha
109H	132x25 chars	16	Alpha
10AH	132x43 chars	16	Alpha
10BH	132x50 chars	16	Alpha
10CH	132x60 chars	16	Alpha
10DH	320x200 pixels	32,768	Graphics
10EH	320x200 pixels	65,536	Graphics
10FH	320x200 pixels	16,777,216	Graphics
110H	640x480 pixels	32,768	Graphics
111H	640x480 pixels	65,536	Graphics
112H	640x480 pixels	16,777,216	Graphics
113H	800x600 pixels	32,768	Graphics
114H	800x600 pixels	65,536	Graphics
115H	800x600 pixels	16,777,216	Graphics
116H	1024x788 pixels	32,768	Graphics
117H	1024x768 pixels	65,536	Graphics
118H	1024x768 pixels	16,777,216	Graphics
119H	1280x1024 pixels	32,768	Graphics
11AH	1280x1024 pixels	65,536	Graphics
11BH	1280x1024 pixels	16,777,216	Graphics

As shown in Fig. 6, when an image is displayed on a computer display, it is "overscanned". That is, the displayed video buffer data occupies less than the entire drivable screen size. The drivable screen size is determined by the total amount of video memory and the

operative video display characteristics. The width of the overscan border that can be used for a complementary user interface depends on the amount of the horizontal overscan 52 reduced by the horizontal blanking 54 and the amount of the vertical overscan 53 reduced by the vertical blanking 55.

5 In one embodiment, only a border at the bottom of the standard display area is used to support a complementary user interface. Consequently, only the vertical control parameters for the cathode ray tube (CRT) controller, shown as Control Registers (CRs) 6H, 16H, 11H, 10H, 12H and 15H in Fig. 4 need to be adjusted. These parameters and others are shown in Table 3 below:

10 **TABLE 3: VERTICAL TIMING PARAMETERS FOR CR PROGRAMMING.**

Register	Name	Description
6H	Vertical Total	Value = (total number of scan lines per frame) – 2 The high-order bits of this value are stored in the overflow registers.
7H	Overflow	High-order bits from other CR registers.
10H	Vertical Retrace Start	Scan line at which vertical retrace starts. The high-order bits of this value are stored in the overflow registers.
11H	Vertical Retrace End	Only the low-order 4 bits of the actual Vertical Retrace End value are stored. (Bit 7 is set to 1 to write-protect registers 0 through 7.)
12H	Vertical Display End	Scan line at which display on the screen ends. The high-order bits of this value are stored in the overflow registers.
15H	Start Vertical Blank	Scan line at which vertical blanking starts. The high-order bits of this value are stored in the overflow registers.
16H	End Vertical Blank	Scan line at which vertical blanking ends. The high order bits of this value are stored in the overflow registers.
59H-5AH	Linear Address Window Position	Linear address window position in 32-bit CPU address space.

15 In the standard 640x480 graphics mode, the nominal horizontal scan rate is 31.5 KHz (31,500 times per second) with a vertical scan rate of 60 Hz (60 frames per second). So the number of lines in one frame is 31,500/60, or 525. Because only 480 lines of data need to be displayed, there are 525-480, or 45, lines available for vertical overscan. Leaving a more than adequate margin for retrace, which requires only 2 lines worth of time, the preferred embodiment uses 25 lines for the alternate display. Thus the additional 18 unused but available lines may be
20 used to increase the size of the native operating system desktop to some non-standard size while still allowing two lines for retrace, or may be left blank, or may be used for one or more additional alternate parallel user interface displays. Similarly, the 1024x768 graphics mode may

have a nominal horizontal scan rate of 68.7 KHz with a vertical scan rate of 85 Hz which computes to 808 lines per frame or 40 lines available for vertical overscan. By modifying the vertical scan rate down to 60 Hz, the frame size increases to 1145 lines which includes 377 lines available for vertical overscan.

2. Modifying the Video Display Area to Support a Complementary GUI

The information display methods of an embodiment of the present invention that uses the physical overscan area to increase display screen real estate can be achieved by providing three capabilities:

- (1) to address and modify the visible resolution of the video display system such that portions of the overscan area are made visible as shown in Fig. 6,
- (2) to address and modify the video display contents for the visible portion of the overscan area, and
- (3) to provide an application programming interface (API) or other mechanism to allow applications to implement this functionality.

Fig. 7, and the additional details and steps provided in Fig.'s 8-13, provides example flow diagrams of an implementation of an embodiment of the present invention that meets the capabilities described above. The environment for this example implementation is a standard Microsoft Windows 95™ operating environment, using Microsoft Visual C and Microsoft MASM with Microsoft's standard platform Software Developer's Kit (SDK) and Device Driver Kit (DDK) for the development platform. One skilled in the art will recognize that other embodiments can perform on other other platforms and within other environments. For example, embodiments could be implemented within any graphical interface environment, such as X-Windows, OSF Motif, Apple Macintosh OS, a Java OS, and others in which similar video standards (VGA, SVGA, XGA, SXGA, UXGA, 8514/A) are practiced. The reference books *PC Video Systems* by Richard Wilton, published by Microsoft Press and *Programmer's Guide to the EGA, VGA, and Super VGA Cards* by Richard F. Ferrano, published by Addison

Wesley, herein incorporated by reference in their entirety, provide more than adequate background information to implement an embodiment in a Windows environment.

As noted earlier, the methods and systems of the present invention also provide other techniques, such as emulation mode, for the alternate display content controller to effectively increase the size of the display area available to parallel user interfaces, by sharing the usable display area between the native GUI and the parallel user interfaces . Emulation mode operates by either effectively shrinking down the portion of the display area allocated to the primary GUI, or by effectively increasing the resolution to a standard or non-standard resolution and utilizing the increase without offering any of the increase to the primary GUI. Emulation mode, as discussed in detail with respect to Figure 14, provides hooks into the video driver and controls what resolution and portion of the screen is allocated to the primary GUI and what is allocated to the parallel GUIs. Note that, regardless of whether overscan techniques are used to increase the displayable area, emulation mode can be used to share the display area between a primary GUI and one or more parallel GUIs.

If the alternate display content controller determines that neither overscan techniques nor emulation mode can be used to display a complementary GUI, then it attempts to use a standard windowed mode provided by the native operating system or primary GUI.

In summary, the alternate display content controller determines how to increase the display area to utilize a complementary GUI, either by increasing the addressable area of the display (e.g., using the overscan area or by using emulation mode and increasing the resolution) or by decreasing the portion of the display usable by the primary GUI, such that remaining display area can be used by one or more complementary GUIs. Use of the overscan area is not automatic – the hardware and software system needs to be accessible to some degree in order to do this (either by knowledge of the video driver and hardware or by a series of heuristics). Several mechanisms can be used to determine whether an overscan technique can be used and are discussed in detail below. If no overscan techniques are usable in a particular video display scenerio, then the alternate display content controller determines whether an “emulation” mode can be used, which shares the resolution of the video display between the primary and any parallel (complementary) GUIs, effectively creating an accessible overscan area.

2.1 Techniques for Extending the Display Area into the Physical Overscan Area (Overscan Mode)

Referring now in particular to Fig. 7, upon initialization, the program determines the screen borders to be accessed in Identify Borders To Display, step 106, based on user preferences and program configuration, and determines, as necessary, whether sufficient video memory exists to make the necessary display changes in the overscan area, step 106. For example, if the screen is currently set to a 1024x768 resolution at 16-bits-per-pixel, and the program is to include four graphical interfaces, one on each edge, with each bar 20 pixels deep, the program must check that video memory is greater than 1.7 MB (required number of bytes = pixels width * Bits Per Pixel * Pixels Height). This calculation is needed only when one or more bars may be displayed on the overscan screen. If the calculation fails to determine that sufficient video memory exists to display the bar or bars in the overscan area, the program proceeds to run in emulation mode. At Identify Display Type, step 102, the program attempts to determine the display type and current location in memory used by the display driver, in order to determine the size and locations of any display modifications to be made, *e.g.*, to the size and location of overscan area(s) to be used.

As described in further detail in Fig. 8, the program first queries the hardware registry in Query Hardware Registry, step 131, to attempt to determine the registered display type. If successful, the program then determines compatibility information in Display Type Supported, step 135, to verify that the program supports that display type and determine memory allocation information.

If the hardware registry information is unavailable, as determined in step 131, or the display type determined in step 131 is unsupported as determined by step 104, the program may use an alternate approach, shown as subroutine Query hardware, steps 135 in Fig. 8, to query the BIOS, in step 134, and the video chipset 66, in step 136, for similar information as described immediately below.

If the BIOS is to be accessed in step 134, physical memory is first allocated in Allocate Physical Memory, step 132, and accessed using Microsoft's DPMI (DOS Protected

Mode Interface) to map it to the linear memory address in which the BIOS resides. It uses DPMI to assign BIOS linear address to physical memory, step 133.

Thereafter, the program queries the BIOS in Read BIOS block, Search for VGA/XVA type and manufacturer ID, step 134. If successful, the driver and chipset are then further queried to determine the display type and memory location in Query driver/chipset for exact chipset, step 136.

If the compatibility information does not indicate a standard VGA, SVGA, XGA, SXGA, UXGA, or 8514/A signature, step 134, this routine returns a failure. If a known chipset manufacturer's identification is found, the driver and/or chipset may be queried with manufacturer-specific routines, step 136, to identify and initialize, as necessary, the specific chipset.

If, in determining the display type, the program identifies a video device driver that is supported by the xSides™ Video Driver Extensions (VDE), the program will use the VDE to implement overscan mode and proceed to run. The xSides™ VDE are extensions that can be implemented by video device driver suppliers to more transparently and congruently support the xSides™ environment. These extensions are described in detail in Appendix E, which is herein incorporated by reference in its entirety.

If, at step 104, the program was unable to finally identify the display type, either because the registry query in step 131 or the hardware query in step 135 was unsuccessful, the program will proceed to run in "emulation" mode.

Returning to Fig. 7, if the program has not already determined that it must proceed in "emulation" mode, it must determine whether it can proceed in "overscan" mode, step 104. There are a number of mechanisms by which this may be done. A set of classes is used, all derived from a common base class corresponding to the below-described VGA-generic technique.

The first mechanism is an implementation of the VGA-generic technique. Using this mechanism, no information specific to a video-card is necessary, other than ensuring VGA support. Using standard application programming interface (API) routines, primary and secondary surfaces are allocated.

Allocation of the primary surface will always be based on the entire screen display. Given the linear address of the allocated primary surface, from which a physical address can be derived, it can be extrapolated that the physical address of the location in video memory immediately adjacent to the primary surface, and therefore immediately below the desktop display, is represented by the sum of the number of bytes of memory used to maintain the primary surface in memory added to the physical address of the primary surface.

Once the physical address of the primary surface is known, the size of the primary surface as represented in video memory can be determined.

For example, the system looks in the CRs for the resolution of the screen, 800 by 600, in terms of number of bits per pixel, or bytes per pixel. Then any data stored in the CR representing any horizontal stride is included. This is the true scan line length.

Next, the physical address of the allocated secondary surface is derived from its linear address. In the case where the allocated secondary surface is, in fact, allocated in the memory space contiguous to the primary surface (the value of the secondary surface physical address is equal to the value of the primary surface physical address plus the size of the primary), the secondary surface is determined to be the location in memory for the overscan display.

If, however, the above is not true and the secondary surface is not contiguous to the primary surface, another approach mechanism is required. For example, a mechanism that “frees” memory from the video device driver to gain contiguous memory by effectively modifying or moving video device driver data may be used. This mechanism may use an interrupt routine to move the driver data transparently.

For example, if the program can identify the Interrupt Descriptor Table (IDT) from the Intel 80386 (or greater and compatible) processors, the program can use the Debug Registers (DRs) to move the driver data found between the primary and secondary display surfaces to a location further down the video memory, making the contiguous memory space available to the program.

The IDT associates each interrupt with a descriptor for the instructions that service the associated event. For example, when a software interrupt (INT 3) is generated (and interrupts are enabled), the Intel processor will suspend what it was currently doing, look up in

the IDT for the appropriate entry (or interrupt vector) for the address of the code to execute to service this interrupt. The code is known as the Interrupt Service Routine (ISR). It will start executing the ISR. When a Return From Interrupt instruction (IRET) is executed by the ISR, the processor will resume what it was doing prior to the interrupt.

Intel 80386 microprocessors (or greater and compatible) provide a set of system registers that are normally used for debugging purposes. These are technically referred to as the Debug Registers (DRs). The DRs allow control over execution of code as well as access over data. The DRs are used in conjunction with exception code. There are four address registers (*i.e.*, Four different locations of code and/or data) (DR0, DR1, DR2, and DR3).

The controlling register (DR7) can be programmed to selectively enable the address registers. In addition, DR7 is used to control the type of access to a memory location that will generate an interrupt. For example, an exception can be raised for reading and or writing a specific memory location or executing a memory location (*i.e.*, Code execution).

Finally, the status register (DR6) is used to detect and determine the debug exception, (*i.e.*, which address register generated the exception). When enabled and the data criterion is met, the x86 processor generates an Interrupt 1 (INT 1).

One example implementation of the alternate display content controller preferably first sets up the IDT to point a new ISR to process INT 1 interrupts. Next, the address of the code to be hooked (or the memory location of data) is programmed into one of the address registers and the appropriate bits within the control register are set. When the x86 processor executes this instruction (or touches the memory location of data), the processor generates an INT 1. The processor will then invoke the Interrupt 1 ISR (as described above.) At this point, the ISR can do almost any kind of processor, code or data manipulation. When complete, the ISR executes an IRET instruction and the processor starts execution after the point of the INT 1 occurrence. The interrupt code has no knowledge of the interruption. This mechanism is used in the example implementation to move the memory address for the video cache and the hardware cursor.

To summarize, the first mechanism for determining whether or not overscan is supported determines how much physical area to allocate for the desktop, allowing adjacent area

for parallel GUI secondary space beyond that to display in the overscan area. The newly allocated area will be the very first block of memory available. If this block immediately follows the primary surface, the physical address will correspond to the value associated with the physical address of the primary surface, plus the size of the primary surface. If that is true, the memory blocks are contiguous, this VGA-generic mechanism can be used to proceed with overscan mode, and the program returns true in step 104 of Fig. 7.

If this first, VGA-generic mechanism cannot be used, the video card and driver name and version information retrieved from the hardware registry or BIOS, as described earlier, is used in conjunction with a look-up table to determine the best alternatives among the remaining mechanisms. The table includes a set of standards keyed to the list of driver names found in the hardware registry. A class object specific to the video chipset is instantiated based, directly or indirectly, on the VGA-generic object.

If the hardware look up does not result in a reliable match, a reliability, or confidence, fudge factor may be used. For example, if the hardware look up determines that an XYZ-brand device of some kind is being used, but the particular XYZ device named is not found in the look up table, a generic model from that chipset manufacturer may often be usable. If no information on the video card is available, the program returns false in step 104 of Fig. 7 and will not proceed in overscan mode.

The next alternative mechanism for determining overscan mode support uses surface overlays. The first step to this approach is to determine if the system will support surface overlays. A call is made to the video driver to determine what features are supported and what other factors are required. If surface overlays are supported, for example, there may be a scaling factor required.

For example, a particular video card in a given machine, using 2 megabytes of video RAM, might support unscaled surface overlays at 1024x768 at 8 bits per pixel, but not at 1024x768 at 16 bits per pixel because the bandwidth of the video card or the speed of the card, coupled with the relatively small amount of video memory would not be sufficient to draw a full width overlay. It is often horizontal scaling that is at issue, preventing the driver from drawing a full width overlay. An overlay is literally an image that is drawn on top of the primary surface.

It is not a secondary surface, which is described above. Typically, the system sends its signal from the video driver to the hardware which in turn merges the two signals together, overlaying the second signal on top of the first.

If a system cannot support unscaled overlays, perhaps because of bandwidth issues or memory issues, this mechanism is not desirable. It is not rejected, but becomes a lower priority alternative. For example, if the scaling factor is below 0.1, then the normal bar can be drawn and it will be clipped closer to the edge. If the scaling factor is more than 10%, another approach mechanism is required

In the next set of alternative mechanisms, a secondary surface is allocated sufficient in size to encompass the normal desktop display area plus the overscan area to be used for display of the overscan bar or bars. Using these mechanisms, the allocated secondary surface does not have to be located contiguous in memory to the primary surface. However, these approaches use more video memory than the others.

The first step is to allocate a secondary surface sufficient in size to contain the video display (the primary surface) plus the overscan area to be used. If the allocation fails, that means that there is not enough video memory to accomplish the task and this set of mechanisms is skipped and the next alternative tried. After the new block of memory is allocated, a timer of very small granularity is used to execute a simple memory copy of the contents of the primary surface onto the appropriate location of this secondary surface. The timer executes the copy at approximately 85 times per second.

Another mechanism for determining overscan mode support is a variant that uses the system page tables to find addresses that correspond to the graphical display interface of the native operating system, such as Windows' GDI. One skilled in the art will recognize that similar methods can be used in systems with other graphical display interfaces to video device drivers. The system page table mechanism queries the system page tables to determine the current GDI surface address, that is, the physical address in the page table for the primary surface. A secondary surface is then created large enough to hold all of what is in the video memory plus the memory required for the overscan area to be displayed. This surface address is then pushed into the system page table and asserted as the GDI surface address.

Thereafter, when GDI reads from or writes to the primary surface through the driver, it actually reads from or writes to a location within the new, larger surface. The program can, subsequently, modify the area of the surface not addressed by GDI. The original primary surface can be de-allocated and the memory usage reclaimed. This mechanism, being more memory-efficient than the previously described mechanism, is the preferred alternative. But this mechanism, modifying the page tables, will not work correctly on a chipset that includes a coprocessor device. If the initial device query reveals that the device does include a coprocessor, this variant mechanism will not be attempted.

Other variations of the above-described mechanisms for determining overscan mode support are handled by derived class objects. For example, the VGA-generic mechanisms may vary when the video card requires more than ten bits to represent the video resolution in the CR. Some instances may require 11 bits. Such registers typically do not use contiguous bytes, but use extension bits to designate the address information for the higher order bits. In this example, the eleventh bit is usually specified in an extended CR register and the extended CR registers are usually chip specific.

Similarly, a variation of the surface overlay mechanism includes a scaling factor, as described above. This alternative is handled in specific implementations through derived class objects and may be the preferred solution in certain situations.

If any of the above-described mechanisms used to determine if overscan mode is supported and subsequently to initialize overscan mode returns a failure, another mode, such as "emulation" mode or "windowed" mode may be used instead.

If the program is to proceed in "overscan" mode, the Controller Registers, or CRs, must first be unlocked, as indicated in Unlock CRTC registers, step 108 in Fig. 7, to make them writeable. The controller registers 6H, 16H, 11H, 10H, 12H and 15H as shown in Fig. 4 and detailed in Table 3, may be accessed through standard input/output ports, using standard inp/outp functions. They are unlocked by clearing bit 7 in controller register 11H.

Addressing of video memory, step 112, is accomplished through one of several means. One is to use the standard VGA 64 Kb "hardware window", moving it along the video memory buffer 66B (Fig. 4) in 64Kb increments as necessary. One example method is to enable

linear addressing by querying the video chipset for the linear window position address, step 138 of Fig. 11. This 32-bit offset in memory allows the program to map the linear memory to a physical address, steps 140 and 142 of Figure 11, that can be manipulated programmatically.

At this point the program can modify the size of the display, step 114 of Fig. 7 to include the border areas. Changing the display resolution to modify the size of the display is shown in detail in Figure 9. In Fig. 9, the routine first checks to determine whether or not the system is running in "emulation" mode, step 144, and, if so, returns true. If not, it then determines whether to reset all registers and values to their original state, effectively returning the display to its original appearance, steps 148-154. The determination is based upon a number of parameters, such as whether the current resolution, step 146, reflects a standard value or previous programmatic manipulation, step 148. If a standard resolution is already set, the variables are reset to include the specified border areas, step 150. If not, the registers are reset to standard values. In both cases the CR registers are adjusted, step 154, to modify the scanned and blanked areas of the display. If the top or side areas are modified, existing video memory is moved accordingly in step 162 of Fig. 10.

If any of the foregoing routines returns a failure, the program may proceed to run in "emulation" mode, step 113 of Fig. 7, if possible, or in windowed mode, step 116 of Fig. 7.

Overscan mode, in the present invention, can be viewed as techniques for adding a secondary GUI by reconfiguring the actual display mode to add a modified, non-standard GUI mode in which the standard display size or resolution has been adjusted to include a secondary display in addition to the primary display. For example, a standard 640x480 display is modified in accordance with techniques of the present invention to become a larger display, one section of which corresponds to the original 640x480 display while another section may correspond to a 640x25 secondary GUI display.

In another embodiment of the present invention system, resources are allocated for a secondary GUI by fooling the video driver into going to larger resolution. This technique guarantees that enough video memory is allocated and unused, since the video driver allocates system resources according to the resolution that the video driver believes it will be operating in. To operate one or more secondary user interfaces in one or more areas of the screen it is

necessary to have the memory within video memory or the frame buffer that is associated with the display location that is contiguously below the primary surface be free and available. By utilizing a series of small routines specific to hardware known to have system resource allocation problems for a secondary user interface, the program may execute such routine whenever resolutions will be switched, initializing the chipset pertinent to that particular routine. If the program finds a routine pertinent to the current particular chipset it will be launched. The routine initializes itself, performs the necessary changes to the driver's video resolution tables, forces a reenab, and sufficient space is subsequently available for one or more secondary user interfaces.

When reenabled, the video driver allocates video memory as needed for the primary display according to the data on the video resolution tables. Therefore, the modified values result in a larger allocation. Once the video driver has allocated memory necessary for the primary surface, the driver will allow no outside modification of the allocated memory. Thus by fooling the driver into believing that it needs to allocate sufficient memory for a resolution exactly x bytes larger than the current resolution where x is the size of one or more secondary user interfaces, the program can be sure that no internal or external use of the allocated memory space can conflict with the secondary user interface.

Fooling the driver into allocating the additional resources can also be done by modifying each instance of the video driver's advertised video mode tables and thus creating a screen size larger than the primary user interface screen size. This technique eliminates the need to prevent the driver from actually shifting into the specified larger resolution and handing the primary user interface a larger display surface resolution. When the video driver validates the new resolution, it will check against a hardware mode table, which has not been updated with the new resolution. (The "hardware mode table," a variant of the aforementioned video resolution tables, is not advertised and not accessible.) This validation, thus, will always fail and the video driver will refuse to shift into that resolution. But, because this technique modified the advertised video mode (resolution) tables early enough in the driver's process, the amount of allocated memory was modified, and memory addresses were set before the failure occurred during the validation process. Subsequently when the CRTC's are modified, in step 114, the

driver has already reserved sufficient memory for one or more secondary user interfaces, which is not available to any other process or purpose.

In yet another embodiment of the present invention, an enveloping driver is installed to sit above the actual (primary) video driver and shims itself in between the hardware abstraction layer and the primary video driver in order to be able to handle all calls to the primary driver. This technique modifies the primary driver and its tables in a much more generic fashion rather than in a chipset specific fashion. The enveloping driver shims into the primary video driver, transparently handling calls back and forth to the primary video driver. The enveloping driver finds the video resolution tables in the primary video driver, which may be in a number of locations within the driver. The enveloping driver modifies the tables (for example, increasing 800x600 to 800x625). A 1024x768 table entry may become, for example, an 1024x800 entry.

Like the previously described technique for fooling the video driver, the primary video driver cannot validate the new resolution and therefore cannot actually change the display resolution setting. As a result, the primary video driver has allocated memory, allocated the cache space, determined memory addresses and moved cache and offscreen buffers as necessary, but is unable to use all of the space allocated, or draw into that space.

2.2 Techniques for Sharing the Display Area (Emulation Mode)

Emulation mode uses a “hooking” mechanism, as shown in Fig. 14, to use and reallocate display areas and driver resources. After the video device driver is identified through the hardware registry or the BIOS, *e.g.*, as described above, certain programming interface entry points into the driver are hooked, such as in step 117, to control parameters passed to and from the driver. When the operating system’s graphical device interface, for example GDI, calls those entry points into the video device driver, the alternate display content controller modifies the parameters being passed to the driver, and/or modifies the values being returned from the driver, thereby controlling the attributes of the display communicated back to the native operating system’s graphical device interface. The program thus “hooks” (or intercepts) calls to the video device driver to and from the graphical device interface.

For example, by hooking the “ReEnable” function in the display driver, at step 117, the program can allocate screen area in different ways in step 119:

(1) In step-up mode, step 121, by intercepting a resolution change request and identifying the next-higher supported screen resolution and passing that higher resolution to the video device driver and when the driver acknowledges the change, intercepting the returned value, which would reflect the new resolution, and actually returning the original requested resolution instead. For example, when GDI requests a change from 640x480 resolution to 800x600 resolution; the program intercepts the request and modifies it to change the video device driver to the next supported resolution higher than 800x600, say 1024x768. The driver will change the screen resolution to 1024x768 and return that new resolution. The program intercepts the return and instead passes the original request, 800x600, to GDI. The video device driver has allocated and displays a 1024x768 area of memory. GDI and the native OS will display the desktop in an 800x600 area of the display, leaving areas on the right and bottom edges of the screen available to the program.

(2) In share mode, step 123, the program intercepts only the return from the video device driver and modifies the value to change the graphical device interface’s understanding of the actual screen resolution. For example, when GDI requests a change from 800x600 resolution to 1024x768 resolution, the program intercepts the returned acknowledgment, subtracting a predetermined amount, for example, 32, before passing the return on to GDI. The video device driver has allocated and displays a 1024x768 area of memory. GDI will display the desktop in an 1024x736 area of the display, leaving an area on the bottom edge of the screen available to the program.

(3) In step-down mode, step 125, the program performs the reverse of step-up mode: that is, the program intercepts a resolution change request; requests the

resolution change, but returns the next lower resolution to the graphical device interface. For example, when GDI requests a change from 640x480 resolution to 800x600 resolution; the program intercepts the request and modifies it to change the video device driver to 800x600. The video device driver will change the screen resolution to 800x600 and returns that new resolution. The program intercepts the return and instead passes a next lower resolution, 640x480 (denying the request), to GDI. The driver has allocated and displays a 800x600 area of memory. GDI and the native OS will display the desktop in an 640x480 area of that display, leaving areas on the right and bottom edges of the screen available to the overscan program.

An alternative to these hooking mechanisms would hook all of the necessary video device driver functions to modify the X,Y offsets used for specific GDI functions. Effectively, this moves the GDI display area within the larger screen display area. This mechanism allows the creation of emulation mode space on the top and left edges by sharing some or all of the space initially created for the bottom and/or right edges.

If the video device driver cannot be hooked, as described above, “emulation” mode cannot be supported and the program will proceed to run in “windowed” mode as described with reference to step 116 of Fig. 7. Windowed mode will use established API routines to the native operating system GUI to run as an “application toolbar” within the standard window display area. Running as a standard application enables the program to take advantage of mechanisms available to all applications on the native OS, such as window creation, docking toolbar interfaces, *etc.*, and allows the program to run under standard conditions.

In summary, the alternate display content controller determines how to increase the display area to utilize a complementary GUI, either by increasing the addressable area of the display (e.g., using the overscan area or by using emulation mode and increasing the resolution) or by decreasing the portion of the display usable by the primary GUI, such that remaining display area can be used by one or more complementary GUIs. If no overscan techniques are usable in a particular video display scenerio, then the alternate display content controller

determines whether an “emulation” mode can be used, which shares the resolution of the video display between the primary and any secondary (complementary) GUIs.

3. Rendering Images to the Modified Display Area

Phase 2 of the example embodiments of the present invention begins by painting the new images into an off-screen buffer, step 118, as is commonly used in the art, and making the contents visible, step 120, as described with respect to Fig. 10.

After determining and initializing the program mode, the program can display data by any of these techniques, as appropriate:

(1) Using standard API calls to render the data to an off-screen buffer, as described in the next section, and then hooking the “BitBlt” function entry point into the display driver in order to modify the offset and size parameters, the program subsequently redirect the BitBlt to the area outside of that which the API believes is onscreen.

(2) Using mechanisms of primary and secondary surface addresses, described above, the program determines the linear addresses for the off-desktop memory location(s) left available to it, and can render directly to those memory locations as shown in steps 158 and 142 of Fig. 11 and step 154 of Fig. 10.

(3) Using video device driver escapes, a standard mechanism within the domain of device driver interfaces, the program can blit directly to the video buffer.

(4) If the program is in “windowed” mode, step 156, the off-screen buffer is painted into the standard window client space, step 166, and made visible, step 164, using generic windowing-system routines.

4. Event Handling In Conjunction With the Modified Video Display Area

A preferred embodiment of the program includes a standard application message loop, step 122, which processes system and user events. An example of a minimum functionality process loop is in Fig. 12. Here the alternate display content controller handles a minimal set of system events, such as painting requests, step 170, system resolution changes, step 172, and activation/deactivation, step 174. Here, too, is where user events, such as key or mouse events, may be handled, step 184, detailed in Fig. 13. System paint messages are handled by painting as appropriate into the off-screen buffer, step 178, and painting the window or display buffer, step 180, as appropriate, as described earlier in Fig. 10. System resolution messages are received whenever the system or user changes the screen or color resolution. The program resets all registers and/or hooks, as appropriate for the current modes, to the correct new values, then changes the display resolution, step 182, as earlier described in Fig. 9, to reflect the new resolution modified. User messages can be ignored when the program is not the active application.

Fig. 13 describes a method of implementing user-input events. In this embodiment, there are three alternative mechanisms used to implement cursor or mouse support so that the user has a pointing device input tool within the alternate display content controller area user interface.

According to one preferred mechanism, GDI's "cliprect" is modified to encompass the bar's display area. That keeps the operating system from clipping the cursor as it moves into the overscan area. This change doesn't necessarily make the cursor visible or provide event feedback to the application, but is the first step.

Some current Windows applications continually reset the cliprect. It is a standard programming procedure to reset the cliprect after use or loss of input focus. Some applications use the cliprect to constrain the mouse to a specific area as may be required by the active application. Whenever the program receives the input focus it reasserts the cliprect, making it large enough for the mouse to travel down into the program display space.

Once the cliprect has been expanded, the mouse can generate messages to the operating system reflecting motion within the expansion area. GDI does not draw the cursor outside what it understands to be its resolution, however, and does not pass "out-of-bounds" event messages on to an application. The preferred program uses a VxD device driver, and related callback functions, to make hardware driver calls at ring zero to monitor the actual physical deltas, or changes, in the mouse position and state. Every mouse position or state change is returned as an event to the program which can graphically represent the position within the display space.

An alternative mechanism avoids the need to expand the cliprect in order to avoid conflict with a class of device drivers that use the cliprect to facilitate virtual display panning. Querying the mouse input device directly the program can determine "delta's", changes in position and state. Whenever the cursor touches the very last row or column of pixels on the standard display, it is constrained there by setting the cliprect to a rectangle comprised of only that last row or column. A "virtual" cursor position is derived from the deltas available from the input device. The actual cursor is hidden and a virtual cursor representation is explicitly displayed at the virtual coordinates to provide accurate feedback to the user. If the virtual coordinates move back onto the desktop from the overscan area, the cliprect is cleared, the virtual representation removed, and the actual cursor restored onto the screen.

A third alternative mechanism creates a transparent window that overlaps the actual Windows desktop display area by a predefined number of pixels, for example, two or four pixels. If the mouse enters that small, transparent area, the program hides the cursor. A cursor image is then displayed within the overscan bar area, at the same X-coordinate but at a Y-coordinate correspondingly offset into the overscan area. If a two-pixel overlap area is used, this method uses a granularity of two. Accordingly, this API-only approach provides only limited vertical granularity. This alternative mechanism assures that all implementations will have some degree of mouse-input support, even when cliprect and input device driver solutions fail.

Similarly, the keyboard input can be trapped whenever the mouse is determined to be within the program's display space. Using standard hooking mechanisms available to the native OS and graphical user interface, or alternatively a hook directly unto the keyboard driver,

key events can be trapped and processed whenever the program determines to do so (*e.g.*, when the user moves the mouse onto the program display space). When the key input is trapped, other applications can be prevented from viewing keyboard events. If the keyboard driver itself is hooked, even GDI does not get the key input events.

Fig. 7 also describes the cleanup mechanisms executed when the program is closed, step 124. The display is reset to the original resolution, step 126, and the CR registers are reset to their original values, step 128, and locked, step 130.

5. Rendering Into and Accessing the Modified Display Area

Various techniques can be used by applications, in addition or alternatively, as appropriate, to render into and access the modified display area once it has been created. As previously discussed, in one embodiment, an API to the functionality of the alternate display content controller is provided to applications to enable them to use graphics primitives that fully function within a display area that potentially extends past the display area originally allocated to the native desktop. Several embodiments of these API are provided to support applications in differing system environments and to achieve different functions. These embodiments allow communications with the modified display area from an application to be as transparent as possible, so that the application does not need to know whether it is communicating to an area allocated to the desktop or to an area outside of that allocated to the desktop.

5.1 Techniques for Rendering to a Modified Display in Windows™ Like Environments

In one embodiment, the alternative display content controller provides an API that intercepts and routes all of the calls to a graphics device interface (GDI) invoked by an application to communicate with the display. For example, in the Windows™ environment, the alternate display content controller intercepts all function calls to the GDI application programming interface (API). The controller determines, based upon the coordinates of the window being written to, whether the call should be forwarded to a display driver that can output to an overscan area (a complementary GUI display driver), or whether the call should be

forwarded to the native graphics device interface. One skilled in the art will recognize that other combinations are possible, such as part processing of the request by the complementary GUI display driver before forwarding the request to the native graphics display driver

Figure 37 is an example block diagram of applications using techniques that intercept native graphics interface library calls. In Figure 37, applications 3701 and 3702 are shown making a call to a function of the API of a graphics device interface after loading the GDI code (GDI 32.DLL). Application 3701 is an application which uses the alternate display content controller API (referred to here, for example, as the "xSides" API) to render using techniques of a complementary GUI into the extended display area (*e.g.*, the display area outside of the native desktop rectangle). Application 3702 is a standard Windows™ application. The alternate display content controller 3703 intercepts the call, and determines whether to forward it to a display driver enabled with the techniques of the present invention 3704, or (/and) to a native display driver (*e.g.*, Windows™ GDI) 3705. One skilled in the art will recognize that this interception technique will work with other graphics device interfaces that are loaded dynamically and that define a documented API, for example, the USER 32.DLL of other Windows™ environments.

Using GDI, this interception technique is accomplished by fooling applications into loading a complementary GUI-enabled graphics device interface library (*e.g.*, xSides GDI) instead of the native operating system graphics device interface library (*e.g.*, GDI). Specifically, upon system initialization, the alternate display content controller renames the native graphics device interface library (*e.g.*, "MS GDI 32.DLL") and names its own graphics device interface library (the overscan-enabled GDI) into the name of the native graphics device interface library (*e.g.*, xSides GDI is renamed "GDI.DLL"). When the alternate display content controller's library loads and initializes, it loads the native graphics device interface library, thereby linking directly into the native GUI capabilities. Thereafter, applications transparently call the alternate display content controller, even if they are only invoking routines of the native graphics interface device library.

When an application makes a function call to the new graphics library (*e.g.*, the GDI.DLL shown as 3703), the library needs to determine whether to invoke the API that is

extended display area-enabled (xGDI) or to invoke the native GDI. One skilled in the art will recognize that there are various ways to make this determination. Using one method, at initialization, an application that uses xGDI registers itself by calling an initialization routine of xGDI, so the alternate display content controller knows which applications actually use the xGDI API. Therefore, xGDI knows when an application that is not xGDI-enabled (*e.g.*, application 3702) is making a function call to the new graphics library, and the call can immediately be forwarded on to the native GDI (*e.g.*, “MS GDI 32.DLL”). When, on the other hand, an application that is xGDI-enabled (*e.g.*, application 3701) makes a function call to the new graphics library, the xGDI routines can determine whether the location referred to in the call (relative to where the pointer was located when the call was made) is within the native desktop rectangle or outside of it in the extended display area.

Using another method, the alternate display content controller may allow an application to write into the extended display area regardless of whether it has “registered” itself with xGDI. For example, an application such as a calculator may be initially launched in the native desktop area and then dragged using a mouse into the extended display area. In this case, the extended display area-enabled API can transparently translate the applications calls relative to new origin coordinates in order to render into the extended display area.

One technique for accomplishing this transparent translation is to intercept every call to the native GDI that causes a repaint or a refresh to occur, translate the coordinates and draw into offscreen memory, and then render the offscreen memory contents into the extended display area. One disadvantage of using this technique is that it consumes more memory and has a small performance hit during each paint/refresh (it draws twice for each call).

A second technique for accomplishing this transparent translation is to translate each native GDI function call to an extended display area-enabled function call and to then translate coordinates in each call. For example, the “CreateWindow” function call of GDI forwards to an “XCreateWindow” function call. One difficulty of using this technique is that the alternate display content controller becomes sensitive to modified versions of the native GDI.

5.2 Techniques to Prevent Obscuring Data

In one embodiment, techniques of the present invention provide a mechanism by which arbitrary rectangular regions of a native desktop display can be reserved for a specific application, allowing the creation and presentation of persistent images that cannot be obscured by any other application. These rectangular regions are called Pixel Masks, because they are masks on the pixels in the region. These techniques are provided via software tools and libraries and complementary documentation, which enable applications developers to build applications with persistent presence on the native desktop.

In an example embodiment implemented in the Windows™ environment, the Pixel Mask software is implemented using a variation of the display driver of the alternate display content controller. Essentially, the display driver is augmented to provide a new feature, that of creating and defining Pixel Masks and authorizing sources. The display driver is augmented by inserting a filter layer between the native operating system's graphics device interface (*e.g.*, Windows™ GDI) and the display driver.

Figure 32 is an example block diagram of an application using pixel mask technology in conjunction with an extended display area-enabled display driver. In Figure 32, the Pixel Mask software is shown residing between the enabled display driver 3202 and the native graphics device interface 3303.

There are two primary parts to the Pixel Mask software: an API that provides a programming interface to the application and a filter driver that intercepts calls from the graphics device interface to the display driver and provides the pixel mask functionality.

The Pixel Mask API provides a set of functions that allows the application program to create and define the Pixel Mask regions, and to identify the authorized bitmap that can be displayed in the Pixel Mask region.

The following functions are defined for the Pixel Mask API:

- Pixelmask_Init initializes the Pixel Mask software. Resources are allocated and initialized, and the software is put into a known state.
- Pixelmask_Uninit de-initializes the Pixel Mask software. Resources are freed, and the software is put into an undefined state.

- Pixelmask_CreateMask creates and defines a pixel mask.
- Pixelmask_DeleteMask deletes a previously created pixel mask.
- Pixelmask_ActivateMask activates an existing pixel mask.
- Pixelmask_DeactivateMask deactivates an existing pixel mask. Once deactivated, the display space is no longer reserved.
- Pixelmask_IdentifySource identifies the display source (for example, a bitmap) that has access to an existing pixel mask. All other attempts to display to the display space within the pixel mask will be clipped at the pixel mask boundaries.

The Pixel Mask Display filter augments the functionality of the display driver by allowing the creation of pixel masks and the identification of authorized display sources that can be displayed in the pixel masks. The display filter will clip all other data that is output into the region covered by the pixel masks. The display filter intercepts calls from the native graphics device interface to the display driver. It hooks in processing before calling the display driver and additional processing after the thread of execution returns from the display driver.

For Windows™ 9x systems, the following display driver functions need to have pre- and post- processing added to support the pixel mask feature.

- BitBlt
- BitmapBits
- DeviceBitmapBits
- ExtTextOut
- Output
- Pixel
- SaveScreenBitmap
- ScanLr
- SetDIBitsToDevice
- StretchBlt
- StretchDIBits
- UpdateColors

Each of these functions will be wrapped by an associated Pixel Mask software function that will, for non-authorized sources, check the specified destination against existing pixel masks, and, if there's an intersection, clip the source data so none is displayed in the region covered by the pixel mask. Authorized sources will be passed through to the display driver function to render data.

One skilled in the art will recognize that, in conjunction with using other software techniques that enable rendering to the extended display areas (areas outside of the native desktop display area), the pixel mask techniques can also provide persistent displays outside the desktop.

6. Additional / Alternative Embodiments

6.1 Additional Embodiments of Overscan Techniques for Modifying the Display Area to Support a Complementary GUI

The following embodiments may be used to modify the size of the display area or the allocation of the display area in order to support a complementary user interface:

1. Utilizing the VESA BIOS Extensions (VBE) in place of the CRT Controller registers (Fig. 5) to determine and/or access the linear window position address, step 138, as necessary.
2. Utilizing the VESA BIOS Extensions (VBE) in place of the CRT Controller registers (Fig. 5) to increase the visible display area as drawn by the CRT.
3. Utilizing API's (application programming interfaces) 62 capable of direct driver and/or hardware manipulation, such as Microsoft's DirectX and/or DirectDraw, in place of the CRT Controller registers and/or direct access to the display buffer.
4. Utilizing API's (applications programming interfaces) 62, such as Microsoft's DirectX and/or DirectDraw, capable of direct driver and/or hardware manipulation, to create a second virtual display surface on the primary display with the same purpose, to display a separate and unobscured graphical user interface.

5. Utilizing modifications to the Shell or Shell tray window component of the operating system 63 in place of the CRT Controller registers and/or DirectX access to the display buffer.

6. Utilizing modifications to the Shell or Shell tray window component of the operating system 63 to create a second virtual display surface on the primary display with the same purpose, to display a separate and unobscured graphical user interface.

7. Building this functionality into the actual video drivers 64 and/or mini-drivers. Microsoft Windows provides support for virtual device drivers, VxDs, and/or System Services which could also directly interface with the hardware and drivers. These could also include an API to provide applications with an interface to the modified display.

8. Incorporating the same functionality, with or without the VGA registers, into the BIOS and providing an API to allow applications an interface to the modified display.

9. Incorporating the same functionality into hardware devices, such as monitor itself, with hardware and/or software interfaces to the CPU.

10. Incorporating the same functionality into overlay devices, either through the video device system or not, overlaying the display to provide applications with an interface to the display.

Embodiments of the present invention do not depend solely upon the ability to change the CRTC's to modify the visible display area. As described, additional mechanisms are provided that define other methods of creating and accessing visible areas of the screen that are outside the dimensions of the desktop accessed by the native operating system's user interface. Various other embodiments of the methods and systems of the present invention also will be apparent to one skilled in the art.

6.2 Using Alternate Display Content Controller to Drive the Native Desktop

Techniques of the present invention may be used to control the desktop *e.g.*, Windows) to easily enable the desktop to operate in virtually any non-standard size limited only by the capability of the display hardware. This may be in combination with parallel graphical

user interface displays or exclusively to maximize the primary operating system desktop display area. This may not require any modification to the operating system.

For example, the visual display area is conventionally defined by the values maintained in the CRTC registers on the chip and available to the driver. The normally displayed area is defined by VGA standards, and subsequently by SVGA standards, to be a preset number of modes, each mode including a particular display resolution which specifies the area of the display in which the desktop can be displayed.

The desktop of a typical native operating system, *e.g.*, Windows, can only be displayed in this area because the operating system does not directly read/write the video memory, rather it uses programming interface calls to the video driver. The video driver simply reads/writes using an address that happens to be in video memory. So this mechanism needs to recognize the value (*e.g.*, address) that the video card and driver assert is available for painting. This value is queried from the registers, modified by specific amounts to increase the display area, and rewritten to the card. In this manner, example embodiments can change the attributes of writable and visible display area without informing the operating system's display interface of the change.

Embodiments of the present invention don't necessarily change the CRTC's to add just to the bottom. Preferably the top is also moved up a little. This keeps the displayed interfaces centered within the drivable display area. For example, rather than just add thirty-two scan lines to the bottom, the top of the display area is moved up by sixteen lines.

6.3 Additional Embodiments for Locating Parallel User Interfaces

One skilled in the art will recognize that any number of parallel GUIs may be positioned in areas not normally considered the conventional overscan area. For example, a secondary GUI may be positioned in a small square exactly in the center of the normal display in order to provide a service required by the particular system and application. In fact, the techniques of reading and rewriting screen display information can be used to maintain the primary GUI information, or portions of it, in an additional memory and selectively on a timed, computed, interactive, or any other basis, replace a portion of the primary GUI with the

secondary GUI such as a pop-up, window, or any other display space. One skilled in the art will recognize that the techniques discussed can be used to effectively position a secondary GUI anywhere on the display screen that is addressable by the alternate display content controller. The controller can also be used to control the relationship between the native (primary) GUI and any secondary GUIs in terms of what is displayed, in what location, at what time.

As a simple example, a security system may require the ability to display information to a user without regard to the status of the computer system and/or require the user to make a selection, such as call for help by clicking on "911?". Embodiments of the present invention could provide a video display buffer in which a portion of the primary GUI interface was continuously recorded and displayed in a secondary GUI for example in the center of the screen. Under non-emergency conditions, the secondary GUI would then be effectively invisible in that the user would not notice anything except the primary GUI.

Under the appropriate emergency conditions, an alarm monitor could cause the secondary GUI to present the "911?" to the user by overwriting the copy of the primary display stored in the secondary GUI memory. Alternatively, a database of photographs may be stored and one recalled in response to an incoming phone call in which caller ID identified a phone number associated with a database photo entry.

In general, embodiments of the present invention may provide one or more secondary user interfaces which may be useful whenever it is more convenient or desirable to control a portion of the total display, either outside the primary display in an unused area such as an overscan area or even in a portion of the primary GUI directly or by time division multiplexing, directly by communication with the video memory, or by bypassing at least a portion of the video memory to create a new video memory. In other words, methods and systems of the present invention may provide one or more secondary user interfaces outside of the control of the native system, such as the native operating system, which controls the primary GUI.

Additional user interfaces may be used for a variety of different purposes. For example, a secondary user interface may be used to provide simultaneous access to the Internet, full motion video, and a conference channel. A secondary user interface may be dedicated to a

local network or multiple secondary user interfaces may provide simultaneous access and data for one or more networks to which a particular computer may be connected.

6.4 Alternative Embodiment Using a Universal Trapping Technique to Create and Manipulate a Modified Display

5 In another aspect of the methods and systems of the present invention, a universal “trapping” technique is provided for modifying the display area and running applications in extended display areas (outside of the native desktop). The new method is “universal” in the sense that, unlike several of the other techniques, it is not sensitive to the particular video card and driver installed in the system. These techniques can be used as an alternative to emulation mode. In essence, the trapping technique operates by shrinking the display area (rectangle) allocated to the native desktop and then dynamically swapping the old size and the new size in response to certain events in the system.

Figure 33 is an example screen display of application windows that are displayed using a universal trapping approach for modifying the display area and rendering outside of the native desktop. The native desktop (with a collection of icons displayed) is shown in display area 3301. Parallel GUIs are shown in display areas 3302-3305, here shown as surrounding the native desktop 3301. One skilled in the art will recognize that any combination of these parallel GUIs may exist, on one or more sides of the native desktop, and that several parallel GUIs may coexist on a side.

20 An example embodiment of the trapping technology shrinks the desktop rectangle, and then dynamically swaps the old size and the new size in response to certain events. For example, whenever the mouse moves, the desktop rectangle must be temporarily restored to full screen, in order for the cursor have full range of motion and move into the areas outside the new size (smaller) inner desktop.

25 In order for the desktop to be resized on the fly, the program must first determine where in memory the desktop rectangle is stored. Experimentation has revealed that Windows™ maintains this information in several locations. These locations are undocumented in both 9x and NT. As discussed below, one embodiment determines these locations by hooking particular

function calls to the native windowing system. There are numerous events that must be intercepted in order to maintain the “illusion” of a smaller desktop. These events are enumerated as follows:

32-bit Ring-3 Hooks:

1. The WNDPROC (window main procedure) of each outside window must be hooked. The hook procedure checks for messages related to size and position of the window (WM_DESTROY, WM_SIZING, WM_WINDOPOSCHANGING, WM_WINDOWPOSCHANGED, WM_SYSCOMMAND), in order to maintain proper placement of the window and size of the new desktop.

2. USER32!GetSystemMetrics. This function is hooked to generate fake return values for SM_CXSCREEN and SM_CYSCREEN.

3. USER32!ChangeDisplaySettings. This function is hooked because the alternate display content controller needs to know when the screen mode is changing.

32-bit Ring-0 Hooks:

4. Context Switches. When context is switched to an outside window (a window outside of the native desktop area), the desktop rectangle is expanded to full screen; when switched to an inside window, the desktop rectangle is set back to its small size.

5. SET_DEVICE_FOCUS. This is a VMM control message that is broadcast from the Virtual Display Driver (VDD) whenever the screen is going to DOS full-screen mode or back. We turn on or off trapping in this case. For NT, a different detection method is needed.

16-bit Ring-3 Hooks:

6. USER!Mouse_Event. All mouse events, including mouse moves, come through this function call. The alternate display content controller needs to intercept mouse moves before they are processed by the system. If the mouse is outside the desktop, the desktop rectangle must be temporarily expanded to include the full screen. After the system finishes processing the mouse event, the desktop rectangle is restored to the size it was before the mouse event occurred.

7. GDI!Escape. The hook for this function checks for the GDI Escape code 39, which is MOUSETRAILS. Presumably this code is sent on each mode change. The main reason for this hook is to react to the screen mode changes not caused by the ChangeDisplaySettings[Ex] (e.g. Direct Draw).

5 9. USER!CopyRect and USER!UnionRect. In addition, the two APIs USER!CopyRect and USER!UnionRect (both 16-bit) are hooked during initialization just prior to calling ChangeDisplaySettings. No settings are actually changed, but as a byproduct, the system calls these two APIs with the desktop rectangle as a parameter. This is how the addresses of where the desktop rectangle are stored are collected. One skilled in the art will recognize that
10 other methods, however, are possible, and for NT, necessary.

Figure 34 is an example block diagram of components of an example embodiment of the trapping technique for modifying the display area. This embodiment consists of four components: a 16-bit DLL (TRAP16.DLL) 3401, a 32-bit DLL (TRAP32.DLL) 3402, a 32-bit EXE (TRAP.EXE) 3403, and a VxD (TRAP.VXD) 3404. These modules communicate with
15 each other in a rather complicated way, as shown in Figure 34. TRAP32.DLL is where most of the action takes place. It is a hybrid of ring-three and ring-zero code. TRAP16.DLL is used for trapping APIs in the 16-bit USER and GDI modules. The VxD is simply a "helper" that brokers kernel-mode calls for TRAP32.DLL. TRAP.EXE is the shell that loads the other modules and creates windows around the edge of the desktop.

20 The trapping architecture supports multiple APIs, each residing in a separate DLL. Figure 35 is an example block diagram of the trapping architecture supporting multiple APIs for different windowing environments. The trapping architecture shown in Figure 35 supports a PixelBar API 3503, a Win32-Specific API 3504, and an Other API 3505. The PixelBar API 3503 supports a current implementation of the extended display area support and
25 allows applications, such as xSides, discussed in the Example Complementary User Interfaces section, to run without modification.

A drawback of using the PixelBar API atop the current embodiment of the trapping architecture is that the PixelBar API was designed to be platform independent, with no knowledge of windows; space created outside the native desktop by the trapping technique on

the other hand, is actually a window. So the application passes raw bitmaps down to the PixelBar API, and the trapping support then turns around and copies them to a window. In effect, every pixel is processed twice.

A WIN32-specific API 3504, eliminates the double buffering. A trapping technique-aware application registers itself with the trapping architecture, and then requests that it be run outside the native desktop. This way, the application is running “natively,” with window dimensions that extend into the outside rectangle, and can write pixels directly to its own window without going through an extra API.

Other APIs 3505 include techniques for other applications, such as ADA-viewers, to communicate with the trapping architecture

As discussed above, particular function calls in the native windowing system are hooked to determine the location of the desktop rectangle and in order to maintain the “illusion” of a smaller desktop. Figure 36 is an example block diagram of the trapping architecture communication using kernel mode hooks.

The hooking mechanism works as follows. The first step is to find the linear address of the interception point. The way this is found depends on what is being hooked.

For 16-bit DLLs, an appy time event is scheduled. When the event is triggered, `_SHELL_LoadLibrary` and `_SHELL_GetProcAddress` are executed, returning a 16:16 address which can then be converted to a linear address via `_SelectorMapFlat`.

For 32-bit DLLs, the client calls `GetProcAddress` from Ring 3 and passes the address down to kernel-mode, or, alternatively, called from Ring 3 via an asynchronous procedure call.

Once the linear address is determined, an INT3 instruction is inserted at that address. This is a one-byte software interrupt (opcode CC). The linear address is also saved in an internal table, along with the byte covered up by the INT3 instruction.

Whenever the INT3 is hit, the VxD gets control (because it was hooked at init time with `Hook_PM_Fault`). The current EIP is checked against a table of breakpoints; if it's not found, the INT3 is assumed to have been placed by another process, and the old INT3 handler is called.

If it is the trapping_system INT3, the hook procedure is called, then the byte replaced by the INT3 is temporarily restored, and execution is resumed at the point of the INT3, and single-step mode is turned on. This will execute one instruction and then trigger an INT1. The INT1 handler will then restore the INT3, turn off single-step mode, and resume execution.

5 The main service provided by the kernel-mode component is the interception of the various API entry points, Windows messages, and, in the case of Windows 9x, VMM Control messages.

These kernel-mode services are provided through IOCTLs issued from user-mode programs via the DeviceIoControl function, as described in the block diagram in Figure 36. This is the standard way for 32-bit user-mode programs to communicate with kernel-mode code, and thus provides some consistency between Windows 9x and NT implementations.

6.5 Alternative Embodiments in Unix Environments to Create and Manipulate a Modified Display

15 Windowing environments other than Windows™ utilize other architectures for creating windows and rendering to them. In Unix type environments, several window systems are used, including those modeled after an architecture known as X-Windows, developed by the Massachusetts Institute of Technology (for example, the MIT developed X11 Server). These environments use an API to create windows and resources for them that are based upon a hierarchy of windows. The desktop background is typically mapped to the root (parent) window, and all other applications that wish to participate as a joint collection are mapped to windows that are children of this root window. This way, the window system knows how to distribute events to the applications that own particular windows.

25 In an X-Windows type environment, the methods and systems of the present invention provide a mechanism for dividing windows between the desktop and between the parallel user interfaces of the complementary GUI. Techniques are provided to split the adapter resources for display of information and to create one or more windowing spaces outside of the normal desktop display.

To accomplish this, the alternate display content controller traps the adapter codes and modifies the drawable screen space used by X-Windows. The alternate display content controller then creates its own spaces using a windowing system specifically engineered to display data in the extended display designated spaces.

In one embodiment, the X11 Server code is modified to allow for N number of "Root" Windows to be presented. This is achieved by modifying the X-Server and changing the drawable area for the "User Root" window, and creating a second "xSides root" window. The alternate display content controller (acting as the X11 server) then traps client requests, and events, and directs information to the correct window based on client attributes. For example, an xSides client's requests have specific characteristics that are designed for the xSides space.

In another embodiment, the alternate display content controller creates N number of displays, each controlled by its own X11 display server. First, a master controller switch is created, which is responsible for the full screen management. Then, the display is divided by the number of servers needed, depending upon how many separate extended display spaces are being used. Note that preferably only one of the servers is accessible by the standard X-Client communication. All other instances are alternate display content controller-specific X11 Servers which accept content from client code that has been enabled to use the alternate display content controller API.

A means to implement the multiple-Root window mechanism described above is illustrated in Table 4.

TABLE 4: EXAMPLE CODE

```

Int FindRoot(x, y)
{
    int i;

    for (i = 2; i > 0; i--)
        if ((x >= WindowTable[i]->drawable.x) &&
            (x < WindowTable[i]->drawable.x + WindowTable[i]->drawable.width) &&
            (y >= WindowTable[i]->drawable.y) &&
            (y < WindowTable[i]->drawable.y + WindowTable[i]->drawable.height))
                return i;
}

```

```

    return -1;
}

5  xEvent*FixEvents(count, xEvents)
    int count;
    xEvent *xEvents;
    {
        int i;
10     int wNum;
        for (i = 0; i < count; i++) {
            if ((xEvents[i].u.u.type == KeyPress) ||
                (xEvents[i].u.u.type == KeyRelease) ||
                (xEvents[i].u.u.type == ButtonPress) ||
15         (xEvents[i].u.u.type == ButtonRelease) ||
                (xEvents[i].u.u.type == MotionNotify)) {
                wNum = FindRoot(xEvents[i].u.keyButtonPointer.rootX,
                               xEvents[i].u.keyButtonPointer.rootY);
                xEvents[i].u.keyButtonPointer.root = WindowTable[wNum]->drawable.id;
20         xEvents[i].u.keyButtonPointer.rootX = WindowTable[wNum]->drawable.x;
                xEvents[i].u.keyButtonPointer.rootY = WindowTable[wNum]->drawable.y;
            }
            else if ((xEvents[i].u.u.type == EnterNotify) ||
                    (xEvents[i].u.u.type == LeaveNotify)) {
25         wNum = FindRoot(xEvents[i].u.keyButtonPointer.rootX,
                               xEvents[i].u.keyButtonPointer.rootY);
                xEvents[i].u.keyButtonPointer.root = WindowTable[wNum]->drawable.id;
                xEvents[i].u.keyButtonPointer.rootX = WindowTable[wNum]->drawable.x;
                xEvents[i].u.keyButtonPointer.rootY = WindowTable[wNum]->drawable.y;
30         }
        }
    }

    return xEvents;
}

```

35

7. Initiating the Alternate Display Content Controller

In another embodiment of the present invention, the launching or initiating of the program may be modified and controlled. For example, alternate display content controller 6 may be launched as a service, as an application, or as a user application. As a service, alternate display content controller 6 may be launched as a service within the registry of utility operating

40 system 5B. The first kind of application is launched in the Run section in the registry, and the user application may be initiated from the Start Up Group within the Start button. Thus,

alternate display content controller 6 may be initiated any time from the first thing after graphics mode is enabled to the very last thing initiated.

Launched as a service, alternate display content controller 6 may be visible shortly after utility operating system 5B such as Windows actually addresses the display, and how soon after depends on where alternate display content controller 6 is put in the order of the things that will be launched as services. It may be possible to put alternate display content controller 6 so that it launches as essentially the first service and thus would launch almost at the same time as the drivers, very, very shortly after the drivers are launched. Accordingly, it is possible to have the screen change from text mode to graphics, draw the colored background, immediately re-display with the overscan addressed and a parallel GUI such as CSNB 2 display the very close to the same time as taskbar. Launched as a run-line application, alternate display content controller 6 may be visible in display space 1 shortly after icons appear.

8. Example Complementary User Interface Support

The following descriptions provide some example user interface functionality that can be implemented using methods and techniques of the present invention. Appendices A, B, C, and D, incorporated herein by reference in their entirety, include descriptions and visuals demonstrating many of these user interfaces, including for example, the xSides™ application environment. The xSides™ application environment (hereinafter “xSides™”) implemented by xSides Corporation provides a complementary user interface, which can coexist using the techniques of the present invention with a native desktop such as Windows 95. It includes, among other capabilities, a cylindrical visualization of a secondary user interface, a Portal feature, and a Web Jump (Network Browser) feature that offers Internet browsing and searching capabilities. The Portal feature can include any type of textual or graphical content envisioned by its implementer. One example use of a portal area, as a personal information manager (PIM), is discussed in detail in Appendix C.

xSides™ also includes the ability to create and execute these interfaces through an application programming interface (API) component. An example xSides™ API is included

as Appendix F, which is herein incorporated by reference in its entirety. The xSides™ API supports the creation and maintenance of a secondary GUI, such as the example cylindrical user interface discussed below with reference to Figures 19-21.

One skilled in the art will recognize that many other user interfaces can be realized by the methods, systems, and techniques of the present invention and that these interfaces may be available in conjunction with one another.

8.1 xSides™ Application Environment Overview

The xSides™ environment is an embodiment of the methods and systems of the present invention. It supports a user interface that is always visible and accessible, technically scalable, able to “overtake” the desktop, merge-able, able to provide highly secure data transmissions, easy to use, and small (<1.5 MB to download). Appendix A, which includes several screen displays, shows examples of some of these capabilities. Other examples of these capabilities and techniques provided by the user interface are provided in Appendices B, which is a product specification for one example release of the xSides™ environment, and Appendix C, which is a product specification for an example PIM.

xSides™ is implemented by software (for example, the alternate display content controller discussed above), that is independent of any underlying systems’ user interface. It resides “below” the operating system and “above” the drivers (if the system architecture is viewed from the drivers up through the application software). The xSides™ software communicates directly to the driver level and adjusts video display parameters. It also allows keyboard and mouse events outside of the primary user interface supported by the native operating system as described in the earlier sections.

The technology can deliver, among other things, Internet content and services, third-party applications, Web browsers, personal Internet portals, advertisements, Web-based client-server applications, including audio and video conferencing, and electronic program guides (EPGs). In addition, in conjunction with the use of underlying streaming technologies on the Internet and technologies that support alternate communication media such as broadband cable networks, television protocols, etc., the xSides™ technology is able to support Web-based

applications on settop boxes and, on the other hand, specific device applications, such as telephones and video and audio conferencing on a generic media such as computer display screen using the Internet. Because the xSides™ Technology enables content and functionality to reside physically outside and be controlled independent of the existing operating systems, such content and functionality do not interfere with and cannot be covered by the operating system or the applications that reside on the desktop. In this manner, the xSides™ technology is able to support a “Web-top” interface as opposed to a simple desktop interface. In addition, because the xSides™ technology can be distributed with a microkernel, cross-platform solutions can be offered without the need to load multiple operating systems on a single computer system. For example, xSides™ can support applications such as WebMail, instant messaging, e-faxing, telephony, music players.

The xSides™ Technology is able to support interactive content and applications in a persistent fashion outside of the operating system because it resides outside of the operating system’s control. Because xSides™ resides within an abstraction layer “below” the operating system and “above” the device drivers, xSides™ can adjust the parameters for the video display system, can increase the number of pixels and scan lines, and can enable keyboard and mouse events within the overscan area. This allows xSides™ to dramatically resize the existing desktop, if desired, “uncovering” the majority of the display area around any or all four sides of the desktop, which can then be used to display complementary content and applications. An application programming interface (“API”) to the xSides™ Technology, for example the API of Appendix F allows developers to rapidly develop applications that take advantage of these unique characteristics of the technology. The technology can potentially address every user of an Internet-enabled computer or TV worldwide. In addition, the proliferation of consumer electronics operating systems (*i.e.*, Microsoft CE) in such devices as portable daily planners and set-top boxes further expands the market opportunity for this technology.

Example products that have used xSides™ Technology are variations of co-branded mini-portals, which reside on the user’s display area and feature the content and applications of partner vendors. These products initially appear on the bottom of a computer screen as a thin cylinder icon (the “control bar”) containing a series of control buttons. The

control bar is comprised of a number of faces, which are called "Sides™," each of which can contain different combinations of content, applications and graphics (hence the name xSides™). The user can easily rotate from one Side™ to the next with mouse clicks to view and access the different content present on a given Side™. The ability to rotate the xSides™ interface to different faces expands the available computer display real estate and allows for compatibility among products licensed to different partners, enabling users to easily view and access whatever content they want. The control buttons can perform a variety of tasks, including launching a Web connection or application, displaying tickers and banners of server-delivered content, or can allow the user to launch functions running in an additional xSides™ display area called the xSides™ Portal.

The xSides™ Portal is an Internet display area which can contain any image or application, including email and instant messaging input and output, calendar and address book information, ISP controls, ad-banners, electronic programming guides and Web-based client-server applications. The Portal may be independent of and co-exist with (above, below, or beside) the xSides™ control bar. In one embodiment, the images and applications are html-based; however, one skilled in the art will recognize that the Portal support can be programmed to display data / content in any programming language or format, such as Java-based content or XML. In each case the Portal support is modified to interpret the content source language of choice. Furthermore, the content source for the portal can come from a remote network such as the Internet, an intranet, or from local device storage, such as a hard disk. The xSides™ Portal may be used, for example, to build personal "desktop" Internet portals. Although in one embodiment preferably only one Portal is displayed in conjunction with an xSides™ control bar (there may be multiple bars on the screen), multiple Portals can be associated with a single side, provided each Portal is accessible through a user interface component such as a button or menu. As mentioned above, Appendix C provides a detailed description of an application that uses the Portal as a personal information management tool (a PIM).

8.2 xSides™ Architecture

In a preferred embodiment, the xSides™ technology is implemented by a distributed architecture comprised of client and server computer systems. Appendix G, which is herein incorporated by reference in its entirety, describes several of the components of this architecture. Programmatic access to the functions of these components can be provided by an application programming interface, for example, the Pixel Bar API of Appendix F.

In one embodiment, the content (the sides) for user control bars is stored on one or more xSides™ servers and users communicate to these servers via network connections. Figure 31 contains an example block diagram of an implementation of the xSides™ architecture. Server computer system 3101 is connected to client computer system 3102 through a set of communication and configuration mechanisms, 3103 and 3104, respectively, which interface to a client side application 3105 responsible for the display of the xSides™ control bar. (Although not shown in Fig. 31, one skilled in the art will recognize that the communication and configuration mechanisms 3103 and 3104 have server-side counterparts, which are components of the server 3106.) One skilled in the art will appreciate that the server computer system 3101 and the client computer system 3102 may in implementation reside in a multiple of distributed or non-distributed layouts, including that an xSides™ server may be distributed over several systems or may reside on the same machine as the client components. One skilled in the art will also appreciate that other configurations and components are possible and may be used to implement the technology described herein.

Referring to Figure 31, the user downloads the partner's content initially from a server machine upon installation of xSides™ on the user's client machine. The content is initially stored within a database or file system, such as database 3107 or file system 3108. Once the xSides™ server machine 3106 sends the content to the xSides™ application 3105, through the communications layer 3103, the client computer system 3102 can store a local copy of the user's control bar and configuration information on local database / file system 3109.

The communications layer 3103 functions to streamline the communications between the client computer system 3102 and the server computer system 3101 and supports the modularized updates of client-side information. Communication is performed preferably using

encrypted markup (e.g., SGML) files, which are sent across the network connection using standard HTTP packet processing. The communications layer 3103 streamlines requests by combining the requests from the various dynamic link libraries (“DLLs”) that handle client-side functions into a single request packet that is sent to an xSides™ server. For example, the sides management functionality that enables users to add and remove sides and the various statistical functions are preferably handled using separate DLLs. When these DLLs need to issue requests, they send them to the communications layer 3103, which combines the requests by placing tags that corresponds to each request in a single packet that is forwarded to the server. The server then deconstructs each packet to process the actual requests. Streamlining the communication in this manner minimizes network traffic and delays.

In addition, the communications layer (client and server portions) enables the ability schedule server communication (ping the server for information) and to schedule the completion of server side tasks on behalf of dependent components on the client side. For example, the Stats/Logging mechanism described below may schedule the updates of server-side logging information on a periodic basis. Also, the components of the client-side xSides™ process, such as the DLLs previously mentioned, can be downloaded at the start of each xSides™ session. Moreover, they can be “hot swapped” to download updated system components in the background. This enables xSides™ to dynamically configure and update itself transparent to the user. One skilled in the art will recognize that the frequency of updates and polling the server for information can be set in any manner – e.g., randomly or explicitly or implicitly by the user or by the application (client- or server-side). In addition, the source and destination for pings and downloads is configurable – thus allowing the configuration of the server-side components to be dynamically configured as well. Appendix G illustrates many of these concepts.

8.2.1 User Configuration

Each xSides™ user is identifiable by a unique global identifier (a “GUID”). GUIDs are used for multiple purposes, including identifying the request and response packets communicated by the communications layer. In addition, since each GUID uniquely identifies

each user, an xSides™ configuration profile can be associated with each user, such that each user can use xSides™ according to the users' preferred configuration, regardless of the physical location of the user and regardless of the machine used by the user to run xSides™. Thus, a user can initiate an xSides™ session from a remote location (such as the users' home computer) and see the same sides (applications) the user sees from the users' normal machine (for example, the users' machine at work). Changes that are made by the user on any machine under the user's GUID are automatically synchronized on the server system, even if multiple instances of xSides™ sessions under the same GUID are running simultaneously.

To provide this functionality, xSides™ provides a User Registration client/server application, preferably implemented as an extractable component such as a DLL, which gathers information from the user and stores it on a server-side file storage mechanism (such as a database). When the user initiates an xSides™ session, the user performs a login, and the user's configuration profile is downloaded (and cached) on the client system. Based upon the configuration profile, xSides™ determines what sides need to be downloaded and cached on the client system, to make the control bar look like what the user would expect. This operation is performed transparently to the user and provides the user's expected environment even if the machine which initiated the request has a version of xSides™ that was installed from a different partner. In brief, the caching mechanisms and general component replacement mechanisms work in conjunction with the merge functionality to provide this configurability.

8.2.2 Merge Function (AllSides)

An important feature of the xSides™ Technology is the ability to "merge" content from multiple partners. Merging is a process in which content from one control bar is merged into another bar. Merge allows users to upgrade their existing xSides™ products to subsequent versions and to add or remove sides (or faces) to a user's control bar at will. An example user interface for explicitly adding and removing sides via merge is shown in the AllSides dialog in Appendix G. Preferably, when a merge takes place, the original distributor's logo and unique content retains its place on the user's bar, and one or more new sides of

information are added. One example implementation of the merge function is included as Appendix D, which is herein incorporated by reference in its entirety.

Essentially, merge enables users to make their xSides™ product a convenient, one-stop destination for all of their favorite content and services. This is not only important and attractive to users, but also to strategic partners who are able to introduce multiple faces, as well as upgrade their users to new applications and functionality over time. Although merge provides product convenience and flexibility for both users and strategic partners, in one preferred embodiment neither the original faces nor the persistent logos on an xSides™ product can be “de-merged,” giving strategic partners additional incentive to distribute the products.

The xSides™ technology also enables users to automatically have the sides of their control bars updated as newer versions become available, for example through the use of a website, *e.g.*, AllSides.com, and a user registration / configuration mechanism. Once a user has installed a side, xSides™ will automatically update the side’s content on a periodic basis (for example, by polling a server machine to determine whether new content is available and downloading the side definition files when it is). Automatic updates are also preferably performed when a partner changes a side and notifies the server machine. As part of these updates, dependent files – such as new component DLLs—can be downloaded to the client machine using the “hot swapping” mechanism described above. In addition, when a user has registered using the User Configuration application and the user logs in to xSides™, xSides™ uses merge technology to create the control bar according to the users configuration profile. This feature is particularly useful when a user travels between different computer systems. Once merged or downloaded, the sides are cached on the client system for efficient access. They can be cached indefinitely or for a period of use or under another expiration-based scheme. In addition, any changes to the user’s configuration profile are posted to the server system.

8.2.3 xSides™ Filtering

In addition to configuration profiles for each user and the ability to add / delete sides dynamically, sides can be filtered by vendor (partner / supplier) and by user class. This capability is useful, for example, for the xSides™ server to determine what to display in a user’s

initial configuration, what a particular user can modify, and for tracking information for a partner. Assuming that the sides for the partners' control bars are stored in a database (other implementations are possible), the database can also maintain stored procedures that correlate a particular user class with the sides available to that user. A vendor in this instance is associated with a list of user classes, each of which are associated with a list of user GUIDs and a list of sides. One skilled in the art will recognize that other organizations for classifying such information are possible and that data structures other than lists or stored procedures may be utilized.

8.2.4 Statistics and Logging Facility

xSides™ also offers a statistics facility and a logging facility, which are described in Appendix G. Preferably, the statistics facility is implemented as a DLL component of the xSides™ application on the client computer system. The purpose of the statistics facility is to gather and record activity and send it to the server computer system to be logged. Once logged, the logging facility uses the data to construct accounting reports and to perform other accounting functions.

The statistics facility records user activity in terms of “clicks” and “impressions.” A click is a mouse click on an xSides™ side or portal; an impression is the amount of time a given area of the xSides™ software is displayed to a user. Thus, if side MyExampleSide is shown to a user, the impression is the time this side is displayed, a click occurs when the user presses a mouse button on a portion of side MyExampleSide. The xSides™ application informs the statistics facility each time a side is displayed (what activity to record) and when a mouse click is trapped (when the activity should be recorded). The statistics DLL prepares a markup string that encodes the recorded data and sends the data on a periodic basis to the server system to be logged. (In one embodiment another DLL is responsible for retrieving the data from the statistics DLL on a periodic basis, *e.g.*, each minute, and for sending the data to the server system.) The markup strings include user and vendor information, thus user activity can be tracked by vendor as well. The logger facility parses the markup strings and enters appropriate data into the database.

In general, the impression time for a side begins when it is first displayed to the user and ends when it is replaced by another display. However, it is possible for the user leave xSides™ running and not be performing any function with it. The statistics facility detects between impressions and mere idle time using a timeout heuristic. Specifically, each impression duration is compared to a timeout value and when it exceeds this timeout value, the impression time is cut off. One skilled in the art will appreciate that other techniques may be used to limit impression duration and to set a minimum for impression duration.

8.2.5 Instant Alert Mechanism

xSides™ also provides a means for partners to send priority messages to their users via a mechanism known as Instant Alerts. The Instant Alert facility is preferably a DLL component and thus communicates with an xSides™ server via the communications layer described above. It can also be automatically updated. The Instant Alert facility allows a partner to send a message to a particular user or to broadcast it to a group (a class) of users. The message content is preferably HTML and is displayed in a browser window on the user's client machine. Each message is markup based with tags that identify the partner, the user GUID etc, and thus each message can be processed using xSides™ communication layer packet transport mechanism. Also, because the messages are markup based and thus contain embedded identifying information, appropriate acknowledgments can be sent back to the server when the message is displayed or received. An overview of flow between the client and server systems in processing Instant Alerts is described in Appendix G.

If a message is used repeatedly, a partner may use a template type message, which includes the ability to name attributes that are filled in when the message is sent. These named attributes function like macros in that their value is computed at the time the message is sent. This value can be the user's GUID, thus providing a unique identifying mechanism for each user. The Instant Alert facility provides tools for creating and managing such template messages. The tool can be form-based or can provide an API for message management.

8.3 Audio and Video Support in the xSides™ Environment

In one embodiment, the xSides™ API provides support for interfacing to other technologies that enable the transmission of audio and video data over broadband cable networks and over the Internet. Support of these technologies allows xSides™ to support applications without having to load an alternate operating system, or multiple operating systems. For example, the API can be used to support two way audio and video applications such as a telephone, a video conferencing application, and other applications that use audio and video streaming technologies, such as those provided by Real Networks Inc. and Broadcom Inc. Also, xSides™ can integrate with the technologies and protocols for the transmission of voice over the Internet and broadband networks (*e.g.*, VoIP, VoDSL, and VoATM). In all these cases, xSides™ presents an API to applications, which hides the underlying technology from applications developers, and allow the developers to present applications in persistent areas on a display screen. These API are compatible with any of the techniques used to modify the display screen and thus can present these persistent applications anywhere on the screen, including outside the desktop in physical overscan space, for example in Portals, in windows on the desktop, or in any combination of the above. Appendix H, herein incorporated by reference in its entirety, shows several example such parallel user interfaces being displayed in conjunction with a native desktop.

In addition, when xSides™ is implemented with a microkernel and is packaged along with the application, the applications can be directly executed on the microkernel and thus execute more efficiently. One skilled in the art will understand that such packaging will enable embedding 2-way communication devices using xSides™ directly in devices that are function specific as opposed to a general purpose computer. In addition, one skilled in the art will recognize that the client applications can run on xSides™ implemented as a microkernel or hosted as services on top of a host OS transparently to the application. These scenerios are demonstrated in Appendix H.

In general, the audio and video streaming technologies over the Internet enable two way voice communication to work as follows: The analog data (such as the voice signals from a telephone or other analog device) are converted from analog to digital and then sent from

a source digital device (such as a source computer system) as digital packets over the network medium (Internet or broadband network). These packets with digital voice data are then reassembled at the destination (such as the receiving computer system), converted from digital to analog data, and sent out directly through a digital device (such as a connected telephone).

- 5 These technologies typically support an API, which hides all of the A/D and D/A conversion and assembling and disassembling of packets.

The xSides™ API marries these technologies to the desktop, by providing an API to the lower level technology APIs to offer application developers a means for providing voice and audio-enabled applications in the xSides™ space. In some cases the API maps one to one with
10 the lower level calls, and in others, it maps one xSides™ API call to several underlying technology calls. In either case, the underlying technical details are transparently provided to the application developer.

8.4 xSides™ Example Cylindrical User Interface

Referring now to Fig. 19, display area 26 includes a parallel GUI 28 according to
15 embodiments of the present invention. Display area 26 may be located anywhere on screen 24S of video monitor 24. For example, with long axis L oriented horizontally display area 26 may be located adjacent edge 24T or edge 24B. Alternatively, with long axis L oriented vertically, display area 26 may be located adjacent edge 24L or edge 24R.

Aspect ratio 34 of parallel GUI 28 is the relationship between dimension 32
20 measured along long axis L and dimension 30 expressed as 34:1 where aspect ratio 34 is determined by equation 36.

$$36 \rightarrow \text{Aspect ratio } 34 = \text{dimension } 32 \div \text{dimension } 30$$

According to a preferred embodiment of the present invention, parallel GUI 28 includes bar 38 surrounded by area 28A. Bar 38 may include one or more containers or
25 cartridges such as cartridge 86 of Fig. 20. Area 28A may be any color; in the example embodiment, area 28A is black. Bar 38 may be composed of separate elements such as title area 40, one or more help areas such as help area 42 and or help area 56, one or more rotators such as rotator 44 and or rotator 48, and one or more buttons such as button 46, button 50, ticker 52 and

button 54. A button may be depressible such as button 46 or non-depressible such as button 40. A depressible button such as button 46 may perform an associated action and display highlighting when selected and clicked on using any conventional pointing device such as mouse 22. A non-depressible button such as button 40 may act as a label and or initiate apparent rotation of the elements of bar 38 to the right of button 40 along with all the associated sound, apparent motion, and highlighting as described below.

During a 'mouse over' condition, that is when a pointer such as arrow 64 is moved over a depressible button such as button 46, the appearance of button frame 62 may be changed such as by changing its color and thus the apparent intensity of emitted light. The change evoked in a button frame such as button frame 62 may be localized to a portion of the button frame such as corner 62A. Preferably, a 'mouse over' condition causes light to apparently emit from the lower left corner of the button frame such as corner 62B.

Clicking on or 'mouse down' condition of a depressible button such as button 46 may evoke apparent movement of the button and or apparent lighting changes adjacent the effected button. Preferably, 'mouse down' of a depressible button such as button 46 causes button 46 to apparently move into bar 38 and an apparent increase of light from behind button frame 62. Apparent motion and light emission changes may be accomplished by any conventional means.

Following a click on or 'mouse down' condition of a depressible button such as button 46 a 'mouse up' condition is initiated thus completing a button selection cycle. A 'mouse up' condition may initiate an action such a hyperlink or launch an application associated with the acting button such as button 46. Additionally, a 'mouse up' condition may cause a button such as button 46 to reverse the apparent motion caused by the prior 'mouse down' condition, thus as in the prior example, button 46 apparently springs back out of bar 38 into alignment with bar 38. At the conclusion of a button selection cycle, a highlighting change of a selected button may also be included. In one embodiment, a post selection highlighting is the same as the earlier described 'mouse over' highlighting and is maintained until another button such as button 54 is selected or some other action within parallel GUI 28 is initiated.

Actuation of a complete button selection cycle on a non-depressible button such as button 50, a title button such as title area 40, or on a rotator such as rotator 44 may initiate rotation about long axis L of the display area. In one embodiment a click of right mouse button 22R initiates rotation of 38 in a first direction D and a click of left mouse button 22L initiates rotation of 38 in a second direction U, opposite first direction D.

Accompanying a complete button selection cycle as described above, sound may be used to enhance the experience and thus heighten the similarity of a virtual metaphor to a real 3-dimensional device. In one embodiment, sound 66 may issue from the computer system; sound 66 may resemble a sound or sounds issued from a real device such as a subtle mechanical click. Any other appropriate sound or sounds may also be used.

A non-depressible button such as button 50 may be used a title button or a placeholder, and thus may not invoke a utility, URL or any other function if subjected to a complete button selection cycle. Accordingly, no highlighting or other special indicia would accompany a 'mouse over' condition of a non-depressible button such as button 50. In an alternate embodiment, a non-depressible button such as button 50 may include the functionality of a rotator such as rotator 44 or 48. Thus a complete button selection cycle on such a non-depressible button would result in the apparent rotation of non-depressible button 50 and all the elements of bar 38 to its right such as ticker 52 and button 60.

Tickers such as ticker 52 may be dynamic reading areas within a cartridge such as cartridge 86 as shown in Fig. 20. Scrolling updateable text such as text 53 can be displayed and the text reading area can also be dynamically linked to launch an application or URL. A ticker such as ticker 52 may be as long as a single button or any combination of multiple buttons. The text such as text 53 that is displayed may be scrolling or otherwise made to move through ticker window 52A. In a currently preferred embodiment of the present invention text enters ticker window 52A at right side 52R and scrolls left, to left side 52L. The scrolling text such as text 53 may repeat in a loop at the end of the text string. Ticker text such as text 53 may be updated locally or over a network. A ticker such as ticker 52 may activate a hyperlink through a network when ticker 52 is clicked on, or subjected to a complete button cycle.

Referring now to Fig. 20, an example of a menu tree that may be displayed and accessed through parallel GUI 28 is shown. Menu 70 includes title bands 72, 74, 76, 78 and 80, which correspond to title area 40, button 46, button 50, ticker 52 and button 54 respectively. Rotators 44 and 48 are represented by bands 82 and 84, respectively. In this example, title area 40 includes 6 containers or cartridges, cartridges 86, 87, 88, 89, 90 and cartridge 91. Many more cartridges and titles may be available; the number of cartridges or titles available may only be limited by the resources of the computer. Cartridges such as cartridge 90 or cartridge 91 may include accessories such as a web browser or media player or any other accessory. Accessories for a cartridge such as cartridge 90 may be installed for use with system software, or they may be components of the software implementing the parallel GUI, or they may be available via a network.

Referring now to Fig. 21, parallel GUI 28 is shown with accessory cartridge 90 visible. Accessory cartridge 90 may include function specific actuators such as fast forward or next track for a CD player. A section of accessory cartridge 90 or any other cartridge selected may also be dedicated to a single function such as web browser 92, to permit the browser to remain visible at all times that parallel GUI software is running.

Cartridges such as cartridges 86-91 may be pre-loaded with links and accessories. Alternatively, the elements or buttons of a cartridge may be blank for loading by a user through a “merge” capability (see Appendix D). User cartridge(s) may include access to applications, documents, files, or network links such as URLs and or embedded functions. Some embedded functions which may be launched from a cartridge may include a browser, an MP3 player, instant messaging, trading notices for marketplace functions, alerts for auction results and or trades, agent checking regarding price comparison searches. User items such as applications, documents, files, or network links may be added to a user button via any conventional method such as copy and paste or drag and drop functions of system software or of any web browser. Preferably, user buttons may be renamed or cleared in any conventional manner.

A parallel GUI such as parallel GUI 28 may also include a help function. Help screens or menus may be implemented in any conventional manner. A map of the contents and organization of bar 38 may be provided in the form of a menu or tree such as menu 70 of Fig. 20.

Menu 70 and other help screens may extend from display area 26 in any conventional manner. In one embodiment, in which menu 70 is visible extending away from edge 26T thus allowing bar 38 to remain visible, actuation of a complete button cycle on a title such as title 87C will initiate rotation of bar 38 to bring cartridge 87 and title 87C to visibility on bar 38.

5 In a one embodiment of the present invention, display area 26 includes 4 preset actuators 94. Activation of a complete button cycle on an actuator such as actuator 96 will rotate bar 38 to a pre-selected position. A user may initially load, change or delete a preset setting associated with an actuator such as actuator 96.

10 The software implementing the parallel GUI may also include a screen saver component such as idle component 96. If parallel GUI 28 is notified that the system software is in idle, rather than blanking display area 26 as in some conventional techniques, parallel GUI 28 may auto rotate through all possible cartridge displays of menu 70. When the system software returns to active mode, bar 38 will automatically return to the last active position prior to idle.

15 If parallel GUI 28 is oriented with a title cartridge, such as cartridge 86 with title 86A visible on title area 40, a complete button cycle of title area 40 as described above may result in apparent rotation of bar 38 and thus display an adjacent cartridge such as cartridge 87 or cartridge 85 (not shown). Title area 40 may also include all buttons and rotators to the right of title area 40 as well. In an alternate embodiment, a complete button cycle of title area 40 changes the visible title such as title 86 and apparently rotates elements of bar 38 to the right of
20 title area 40 such as rotator 44, rotator 48, button 46, button 50, ticker 52 and button 54. The result of changing a cartridge and thus the title visible in title area 40 is that as cartridge 87 is visible, title 87A may be visible as well as a set of it's subordinate titles such as titles 87B, 87C, 87D and 87E. Additional cycling of title area 40 will result in display of additional cartridges and thus additional titles of band 72 such as titles 88A and 89A.

25 If title 89A is visible in band 72, execution of a complete button cycle on rotator 44 corresponding to band 82 will cause apparent rotation of bar 38 at button 46 corresponding to band 74 including everything to the right of button 46. Subsequent button cycles of a rotator such as rotator 44 cause titles which appear on button 46 to sequentially cycle through titles 89B, 89C, 89D, 89E and 89F with a new title appearing after each button cycle. In one preferred

embodiment, a merge function may be included to allow cartridges such as cartridges 86-91 to be added to an existing parallel GUI such as parallel GUI 28. (See Appendix D.) A cartridge such as cartridge 86 may be added or merged with any existing cartridges in a parallel GUI such as parallel GUI 28 using any conventional technique such as copy and paste or drag and drop. A merged cartridge such as cartridge 86 may be added between any two adjacent cartridges such as cartridges 88 and 89. Similarly, existing cartridges may be reordered using a conventional sort function.

New cartridges may be merged or added to an existing parallel GUI from any conventional media such as magnetic storage media, optical storage media, or from network resources such as the Internet, or any local or intranet network. A delete and or a sort function may also be included to permit a user to organize or personalize a bar such as bar 38 in parallel GUI according to their own wishes consistent with the parallel GUI software.

For example, a user may go to a specific Internet site to peruse the applications available to be merged into the parallel GUI. One such application is an application providing access to weather information over the WEB. The user selects the application to be merged, and the parallel GUI automatically determines a set of cartridges provided by the application. The parallel GUI software then merges the determined set of cartridges into the current data structure used to store data on the currently loaded cartridges. One skilled in the art will recognize that any conventional data structure may be used, including arrays, hash tables, linked lists, and trees. Preferably, a data structure that allows easy replacement of entire cartridges (such as cartridges stored as branches of a tree) is used. The parallel GUI software may then update any related data structures whose information depends upon knowledge of the current set of available cartridges.

8.5 Network Browser

Referring again to Fig. 1, in an alternate embodiment of the present invention, the technique of controlling the allocation of display area 1 is used to open a context-sensitive network-browser-2 (CSNB) adjacent but not interfering with operating system desktop 3 and/or parallel graphical user interface 4. A display controller such as alternate display content controller 6 may include CSNB 2 thus permitting the browser to create and control a space for

itself on display 1 which may not be overwritten by utility operating system 5B. The combined controller/browser may be an application running on the computer operating system, or may include an operating system kernel of varying complexity ranging from dependent on the utility operating system for hardware system services to a parallel system independent of the utility operating system and capable of supporting dedicated applications. The alternate display content controller/browser may also include content and operating software such as JAVA delivered over the Internet I or any other LAN. There may also be more than one context sensitive network browser and more than one parallel graphical user interface in addition to the operating system desktop.

Context sensitive interface such as network browser 2 may respond to movement and placement of cursor 1C controlled by a pointing device such as mouse 1M anywhere on display area 1. The generation and control of a cursor across two or more parallel graphical user interfaces was described previously. The location of cursor 1C will trigger CSNB 2 to retrieve appropriate and related network pages such as web page 2A. CSNB 2 may store the last X number of CSNB enabled network addresses for display offline. In a currently preferred embodiment of the present invention, X is ten pages. If a user is examining a saved CSNB enabled page offline, a mouse click on the page or a link on the page will initiate the users dial-up sequence and establish an online connection.

In an alternate embodiment, alternate display content controller 6 may include a browser or search engine. In an alternate embodiment of the present invention, space 2C may include an edit input box 2D. Edit input box 2D may include conventional functionality's such as edit, copy, paste, etc. A user may enter a URL into edit input box 2D using any conventional input device and then select a button to launch or initiate alternate display content controller 6 as a browser. This may be accomplished by using objects and or drivers from utility operating system 5B. Initiating alternate display content controller 6 as a browser would include a simple window to display the URL as a live HTML document with all conventional functionality. By implementing alternate display content controller 6 as a little applet that uses that DLL, it may slide on, or slide off. Thus initiating alternate display content controller 6 as a browser is like a window into the Internet.

Secondly, a user may enter any text into edit input box 2D using any conventional input device and then select a button to launch or initiate alternate display content controller 6 as a search engine. By entering a search string and selecting "search" and enter any string and click on "search" and pass that to any number from one to whatever or existing search engines, and subsequently have the search string acted on by one or more selected search engines and or by alternate display content controller 6 as a search engine. Resulting in multiple different windows appearing in some sort of stacked or cascaded or tiled format, with the different searches within them.

Using alternate display content controller 6 as a search engine or browser, the results or HTML document may be displayed in any overscan area or on the desktop.

Referring now to Fig. 17, a context sensitive network browser such as CSNB 13 may also include a suite of tools such as tools 14 that may or may not have fixed locations on the browser space. Such tools may include but are not limited to e-mail, chat, buddy lists and voice. As shown, spaces such as desktop 14A, web page 14B, secondary GUI 14C and browser 13 may be arranged in any convenient manner.

Although specific embodiments of, and examples for, the present invention are described herein for illustrative purposes, it is not intended that the invention be limited to these embodiments. Equivalent methods, structures, processes, steps, and other modifications within the spirit of the invention fall within the scope of the invention. Also, those skilled in this art will understand how to make changes and modifications to the present invention to meet their specific requirements or conditions. For example, the teachings provided herein of the present invention can be applied to other types of computer systems, including those that control non-integrated display surfaces. In addition, the teachings may be applied to other types of devices that have display surfaces and other organizations of computer operating systems and environments. These and other changes may be made to the invention in light of the above detailed description. Accordingly, the invention is not limited by the disclosure.

CLAIMS

1 1. A method for enabling the display of an image on a video display system
2 in an area outside of a first display area controlled by a computer operating system, the video
3 display system having a total displayable area of which the first display area is a part, the first
4 display area having an original size known to the operating system, comprising:

5 allocating the total displayable area of the video display system to include a
6 second display area by resizing the first display area to a smaller size and allocating the
7 remainder to a second display area that is outside of the first display area;

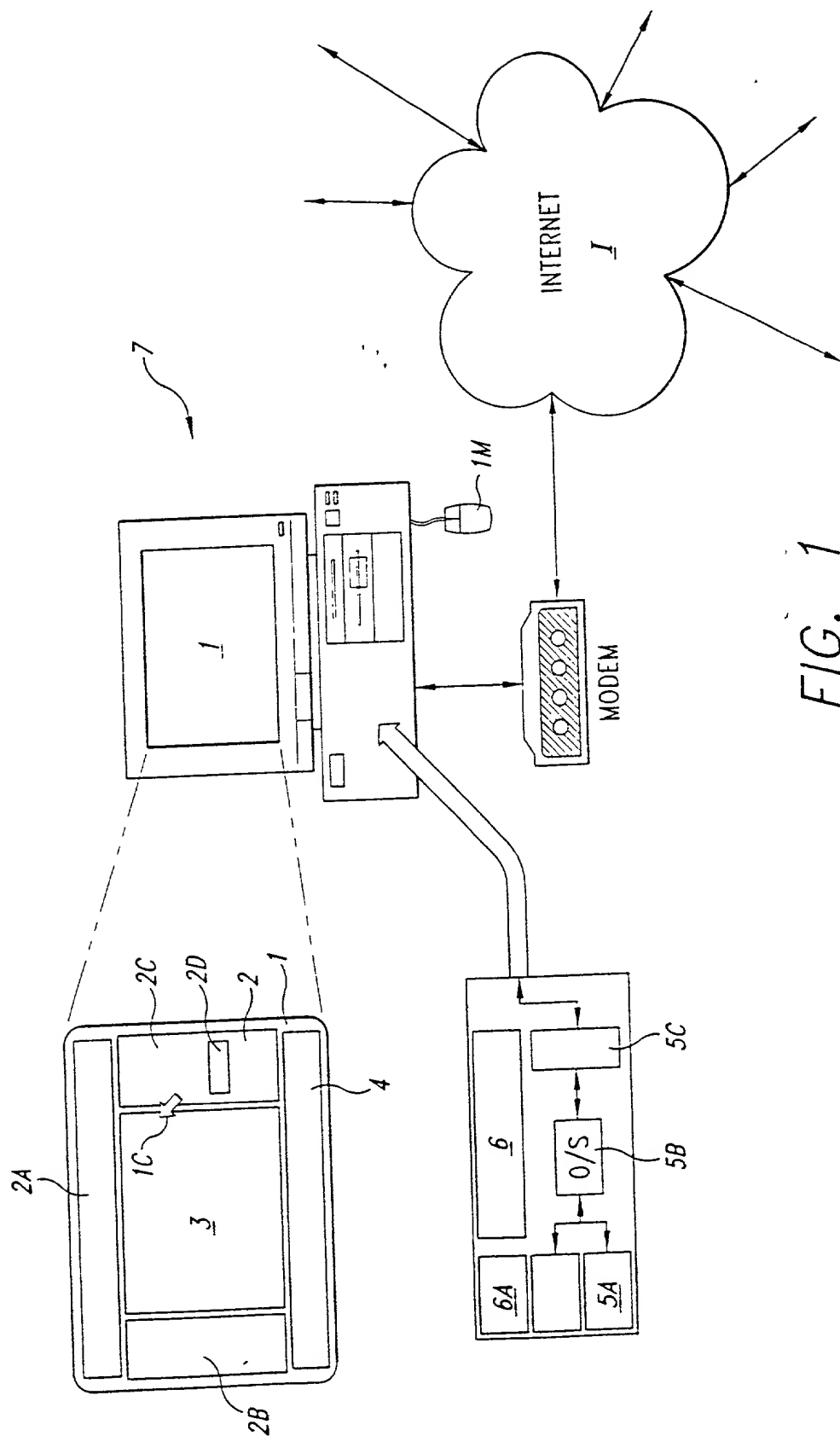
8 intercepting calls that enable an application to obtain input from or output to an
9 input/output device; and

10 transparent to the computer operating system, dynamically reallocating the total
11 display area between the first display area and the second display area by swapping the first
12 display area between the original size and the smaller size to accommodate the intercepted calls,
13 so that the application can render to and obtain input from the first display area and the second
14 display area.

METHOD AND SYSTEM FOR CONTROLLING A COMPLEMENTARY USER
INTERFACE ON A DISPLAY SURFACE

ABSTRACT OF THE DISCLOSURE

An alternate display content controller provides a technique for controlling a video display separately from and in addition to the content displayed on the operating system display surface. Where the display is a computer monitor, the alternate display content controller interacts with the computer utility operating system and hardware drivers to control allocation of display space and create and control one or more parallel graphical user interfaces in addition to the operating system desktop. An alternate display content controller may be incorporated in either hardware or software. As software, an alternate display content controller may be an application running on the computer operating system, or may include an operating system kernel of varying complexity ranging from dependent on the utility operating system for hardware system services to a parallel system independent of the utility operating system and capable of supporting dedicated applications. The alternate display content controller may also include content and operating software delivered over the Internet or any other LAN. The alternate display content controller may also be included in a television decoder/settop box to permit two or more parallel graphical user interfaces to be displayed simultaneously.



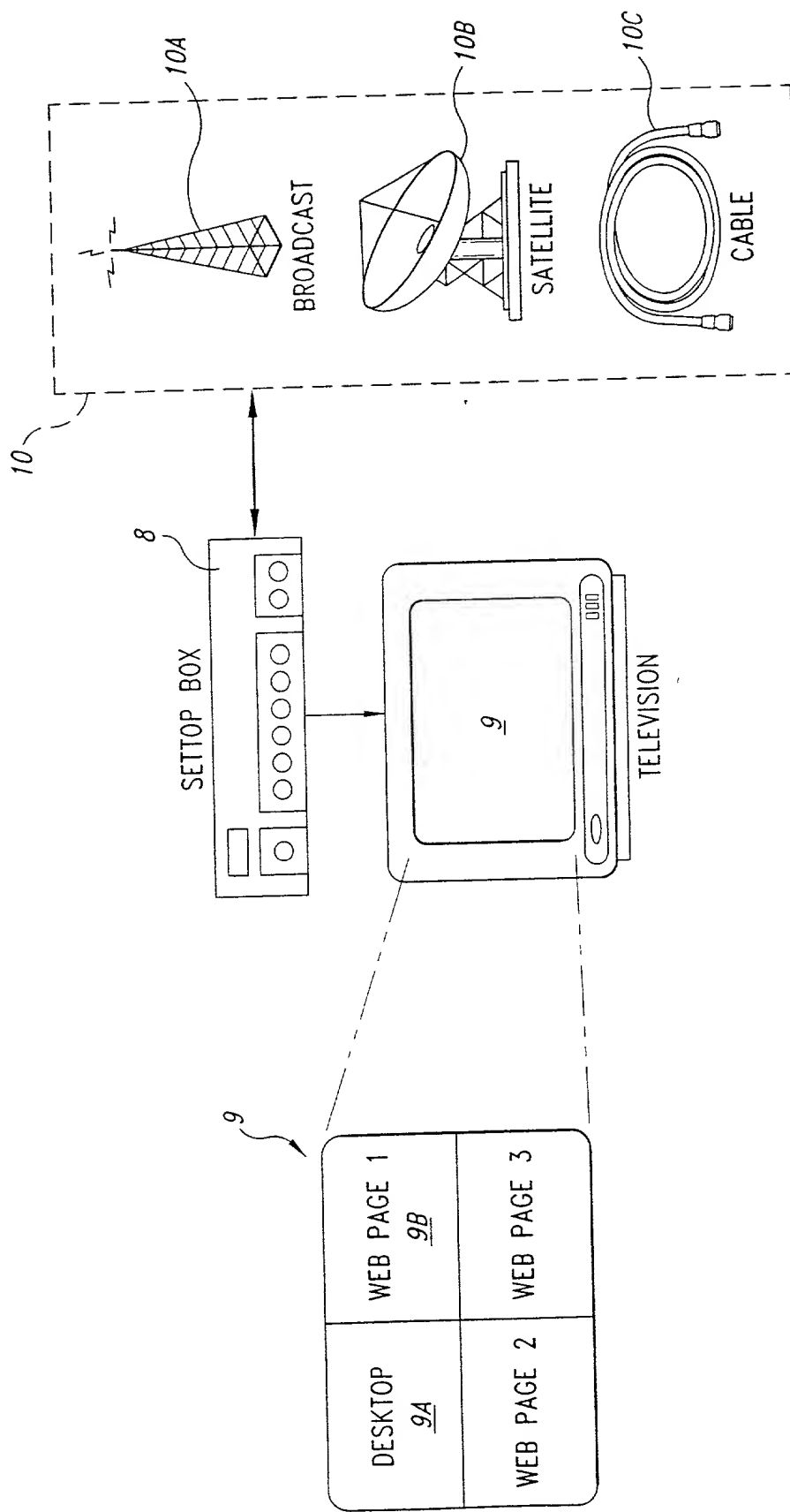


FIG. 2

Figure 1 consists of 12 subplots (a-l) showing the time course of various physiological parameters during a 10-minute period. The x-axis for all plots represents time in minutes, from 0 to 10. The y-axis for each plot represents the parameter being measured. A shaded area indicates the intervention period, and a dashed line represents the baseline level. The parameters are: (a) Heart rate (b/min), (b) Systolic blood pressure (mmHg), (c) Diastolic blood pressure (mmHg), (d) Mean arterial pressure (mmHg), (e) Stroke volume (ml), (f) Cardiac output (l/min), (g) Systemic vascular resistance (dyne/cm²), (h) Pulmonary artery pressure (mmHg), (i) Pulmonary vascular resistance (dyne/cm²), (j) Right ventricular pressure (mmHg), (k) Right ventricular stroke volume (ml), and (l) Right ventricular cardiac output (l/min). The graphs show that during the intervention period, heart rate, stroke volume, and cardiac output decrease, while blood pressure and systemic vascular resistance remain relatively stable. Pulmonary artery pressure, pulmonary vascular resistance, and right ventricular pressure also show a decrease during the intervention period.

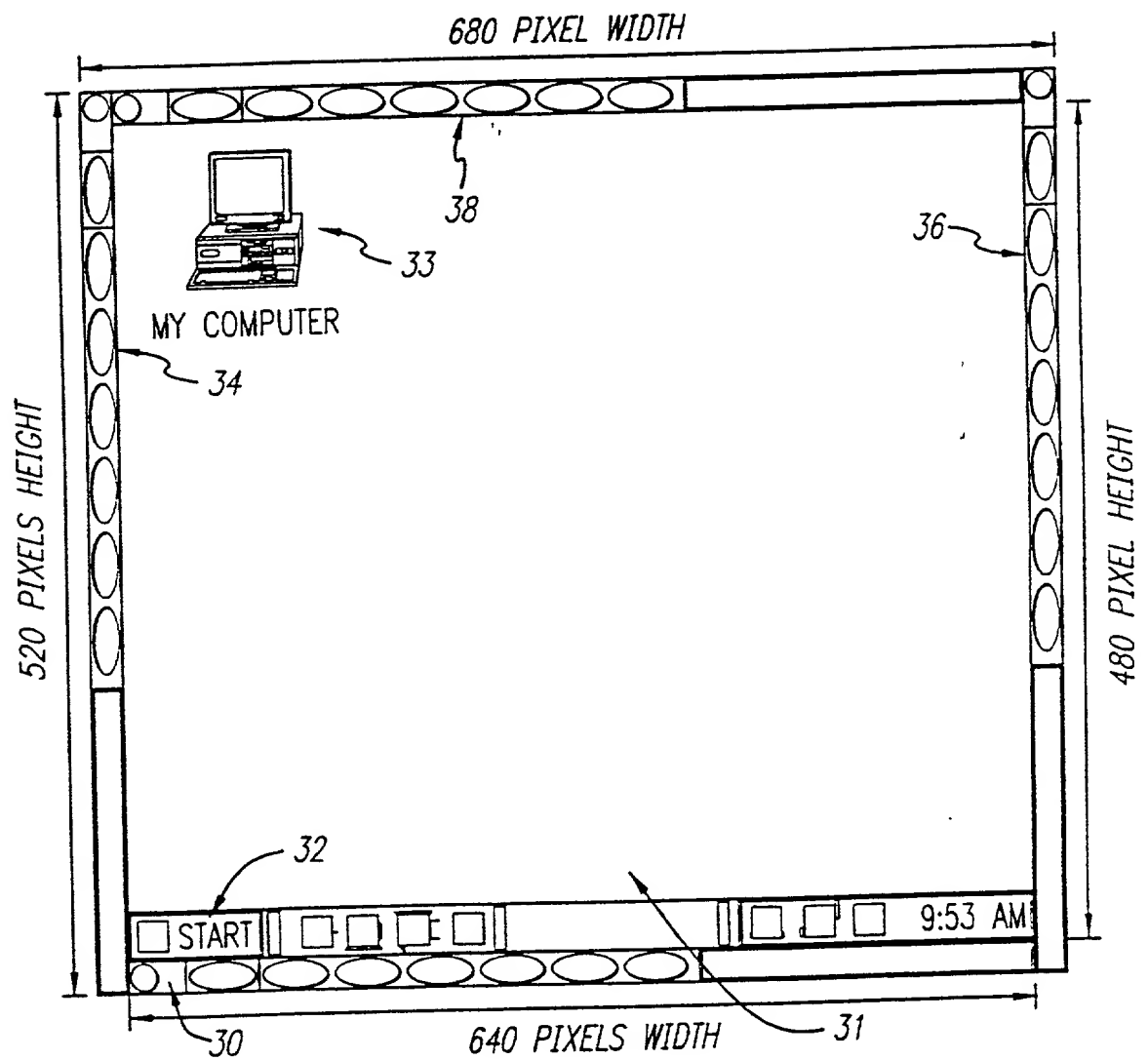
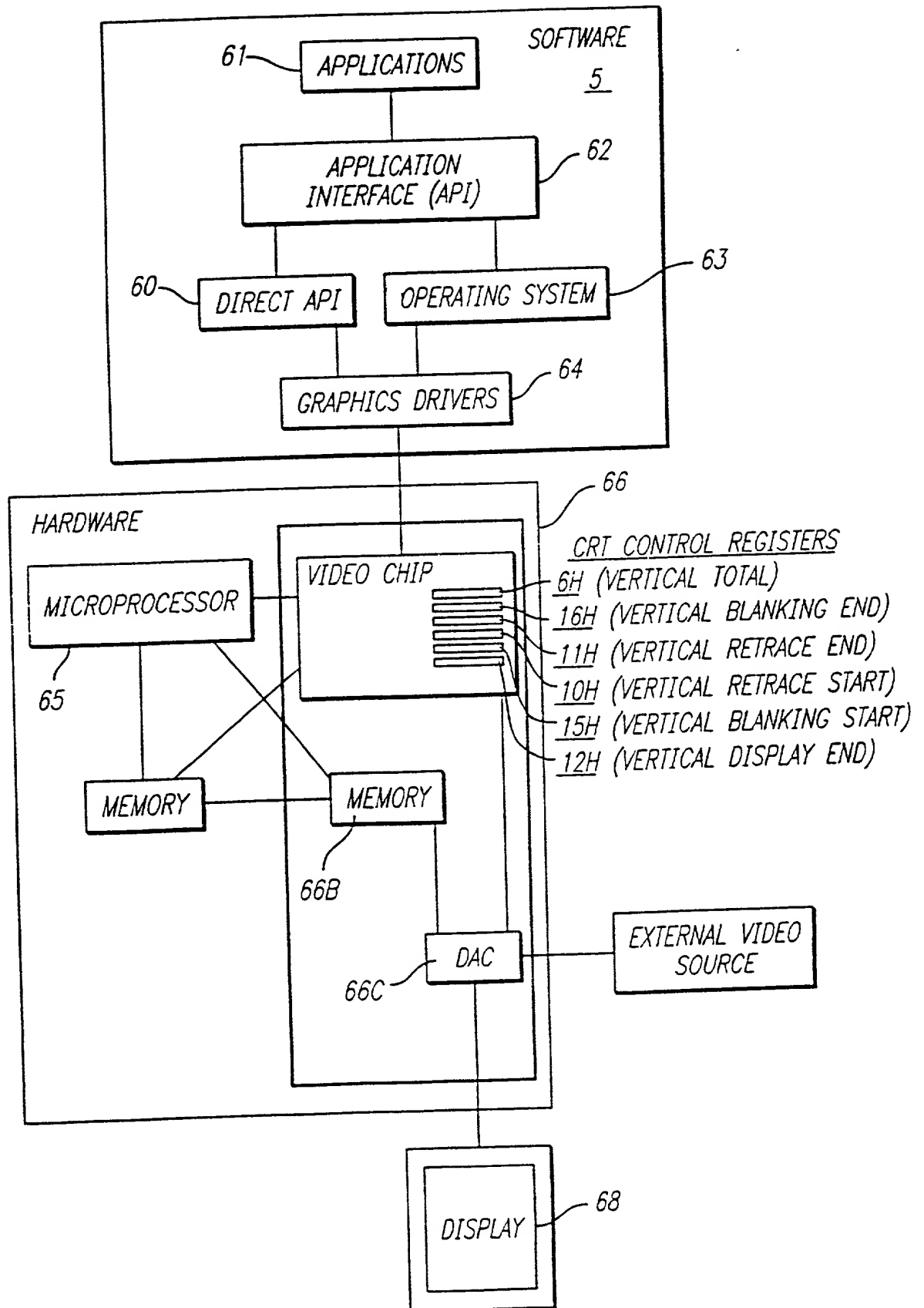


FIG. 4



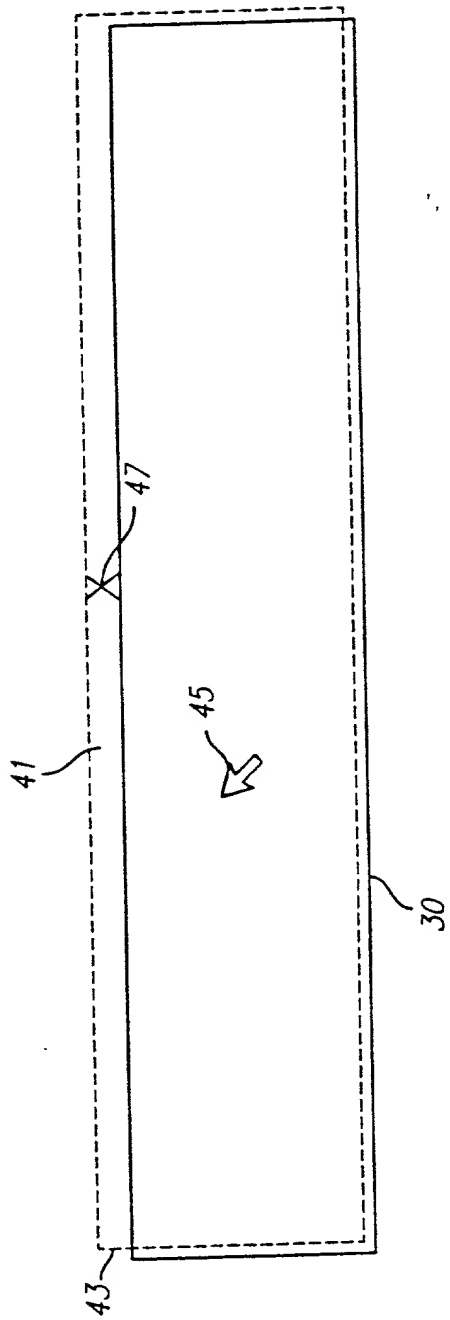


FIG. 5

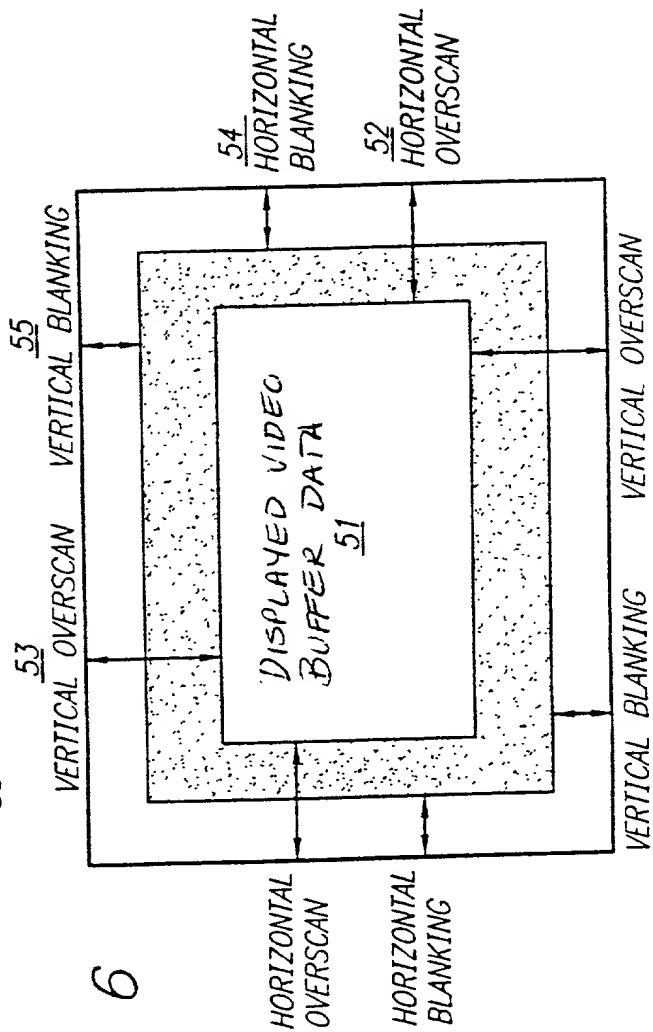


FIG. 6

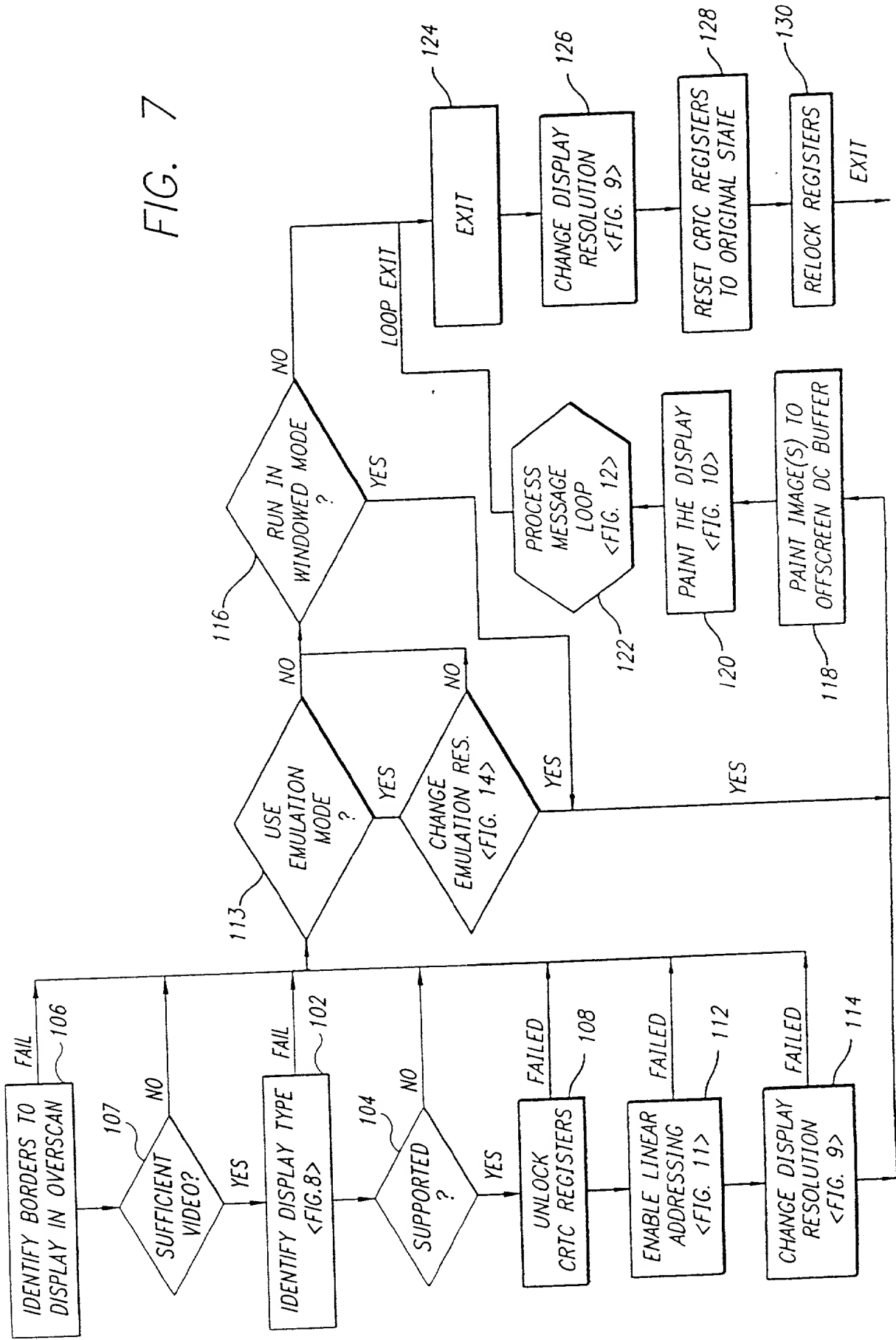


FIG. 7

FIG. 8

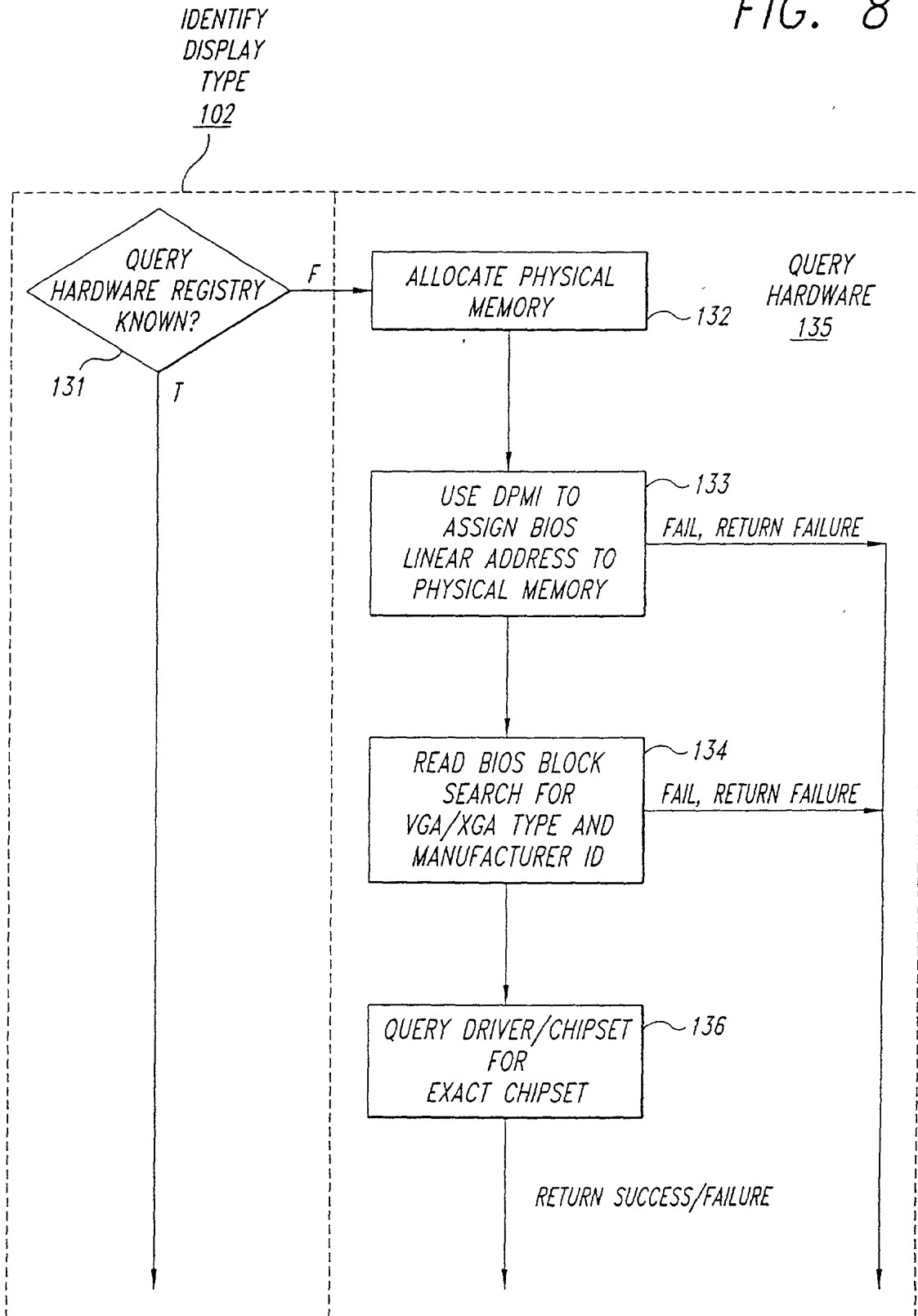
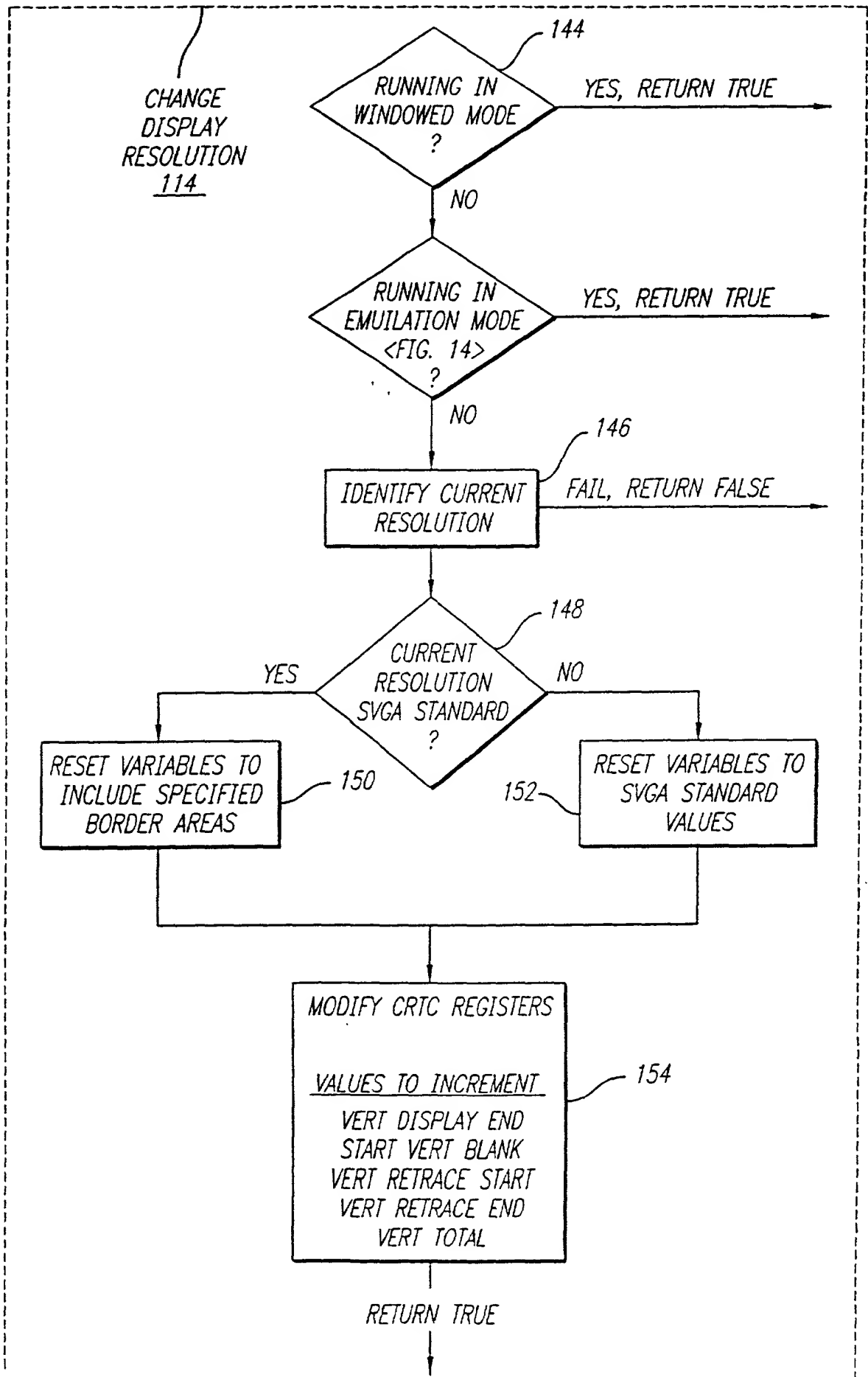


FIG. 9



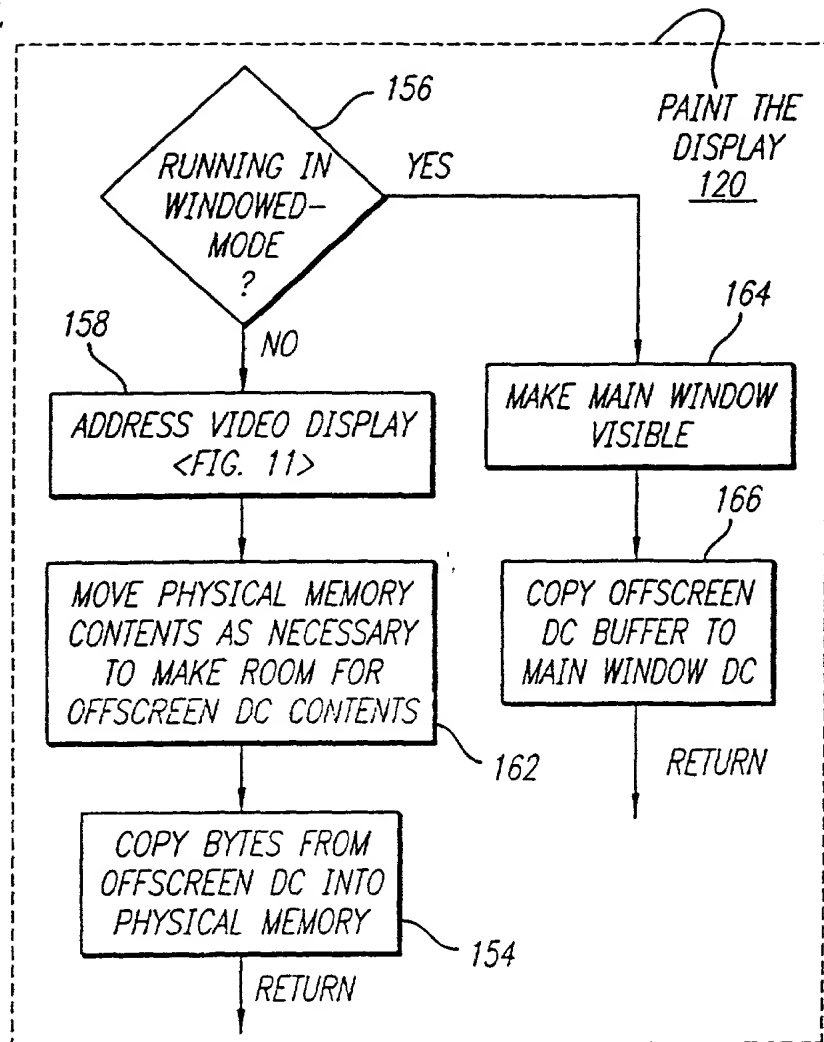


FIG. 10

ENABLE
LINEAR
ADDRESSING
112

FIG. 11

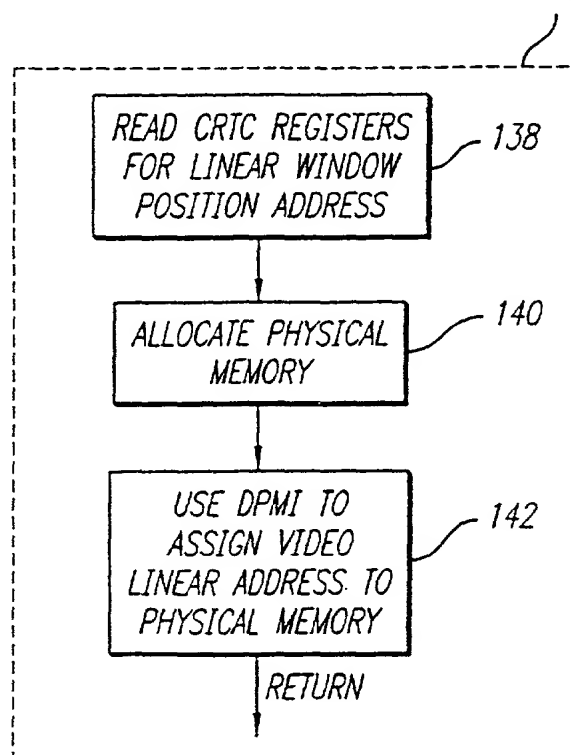


FIG. 12

MESSAGE PROCESS LOOP
USER INTERFACE

122

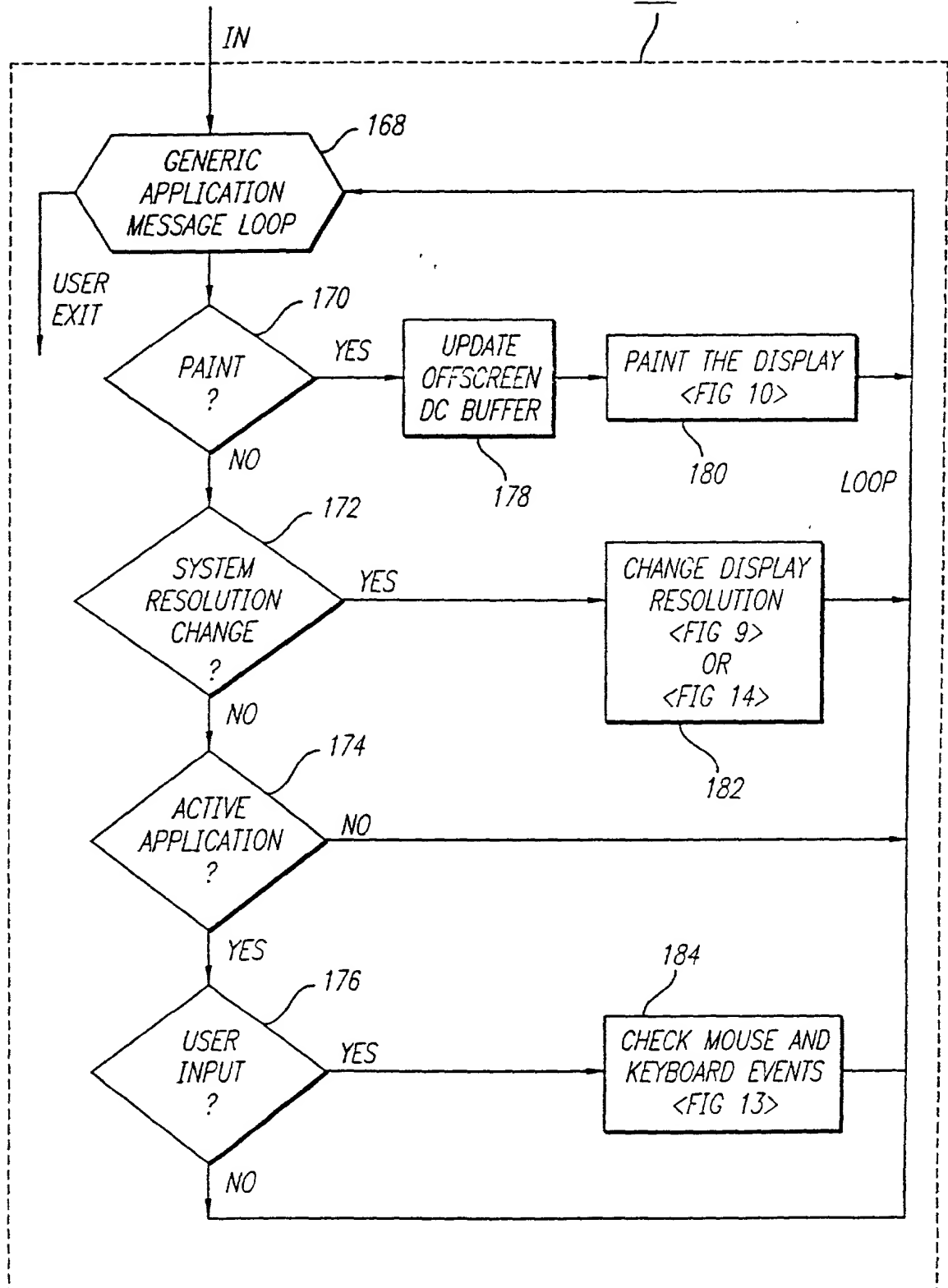


FIG. 13

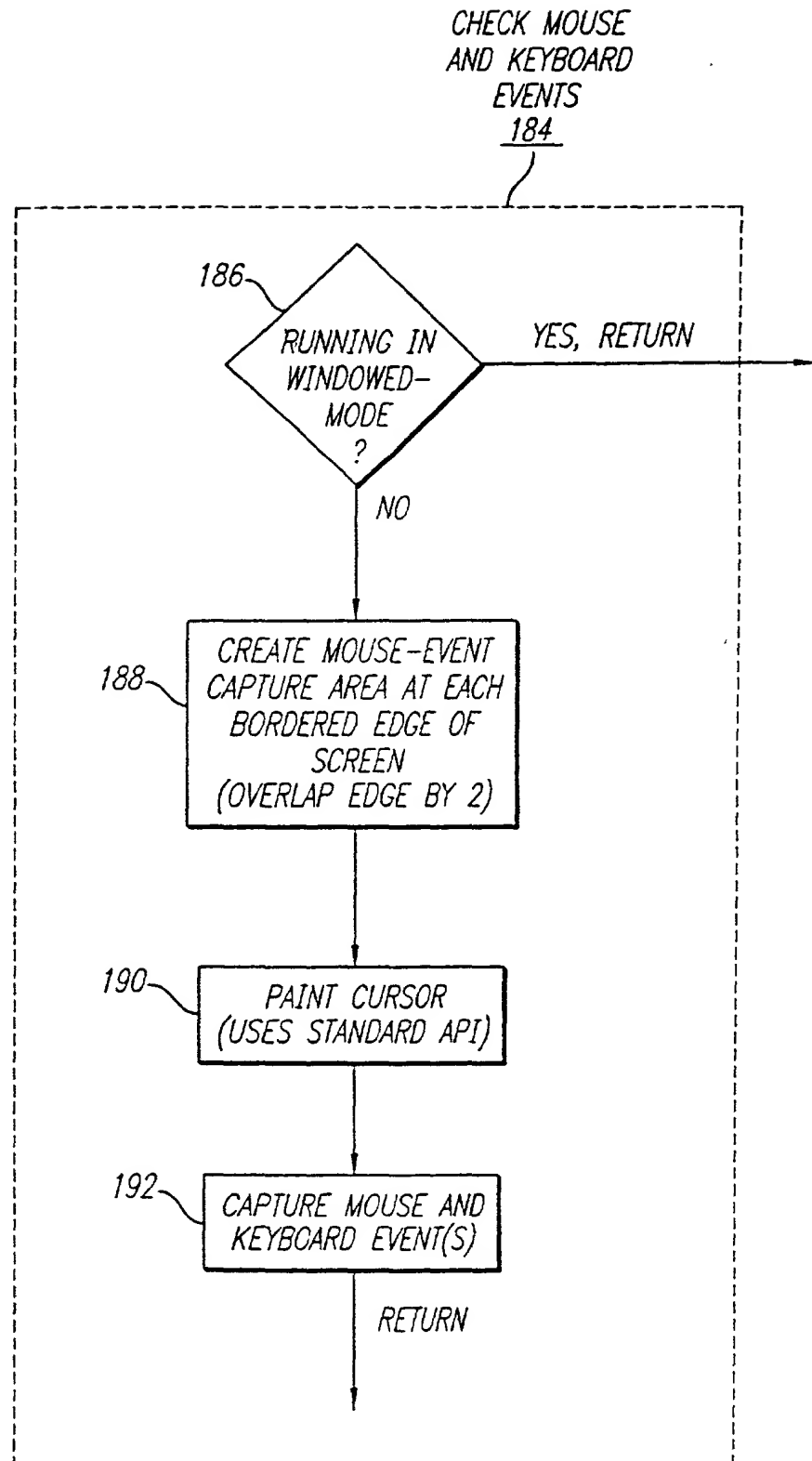


FIG. 14

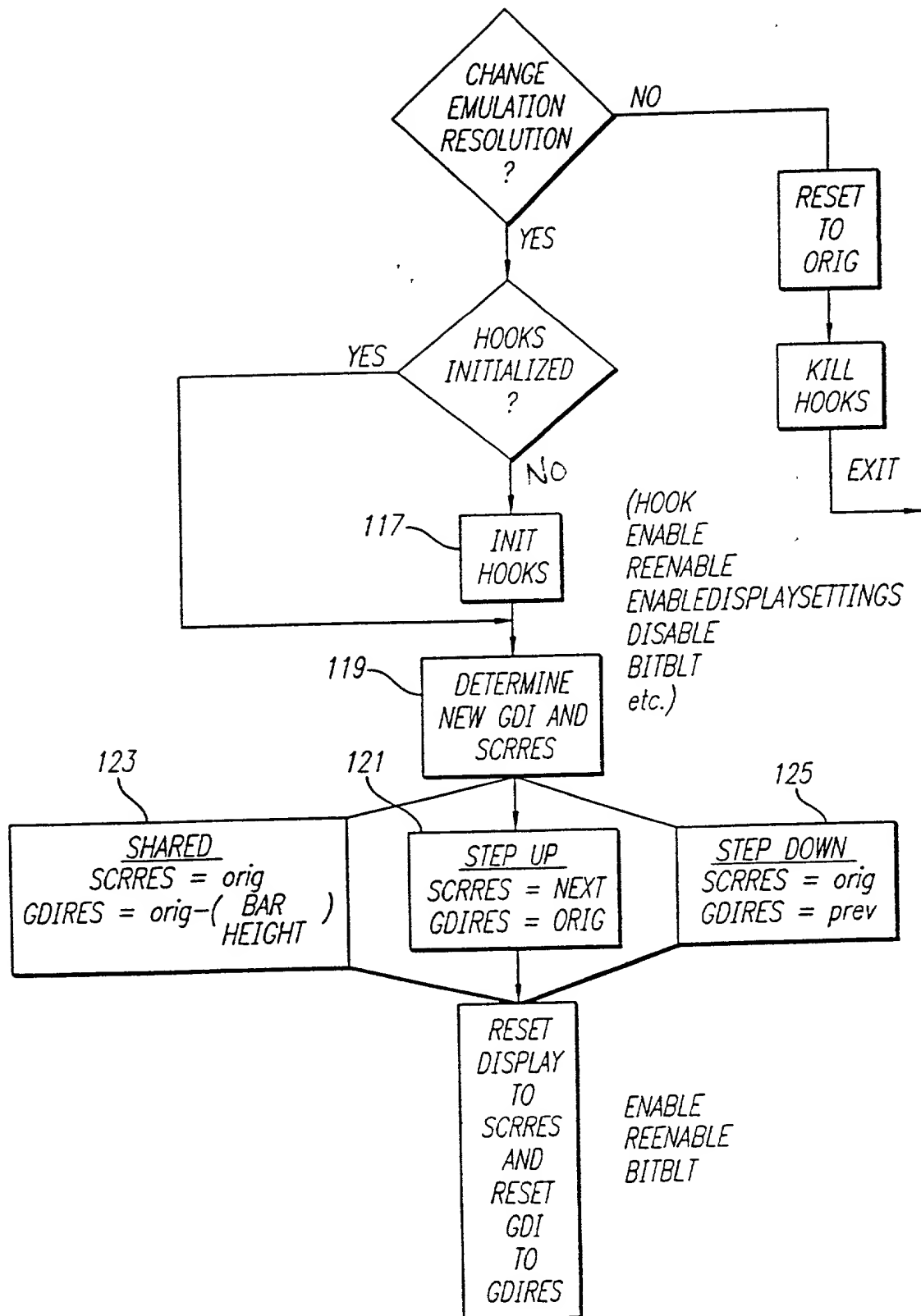


FIG. 15

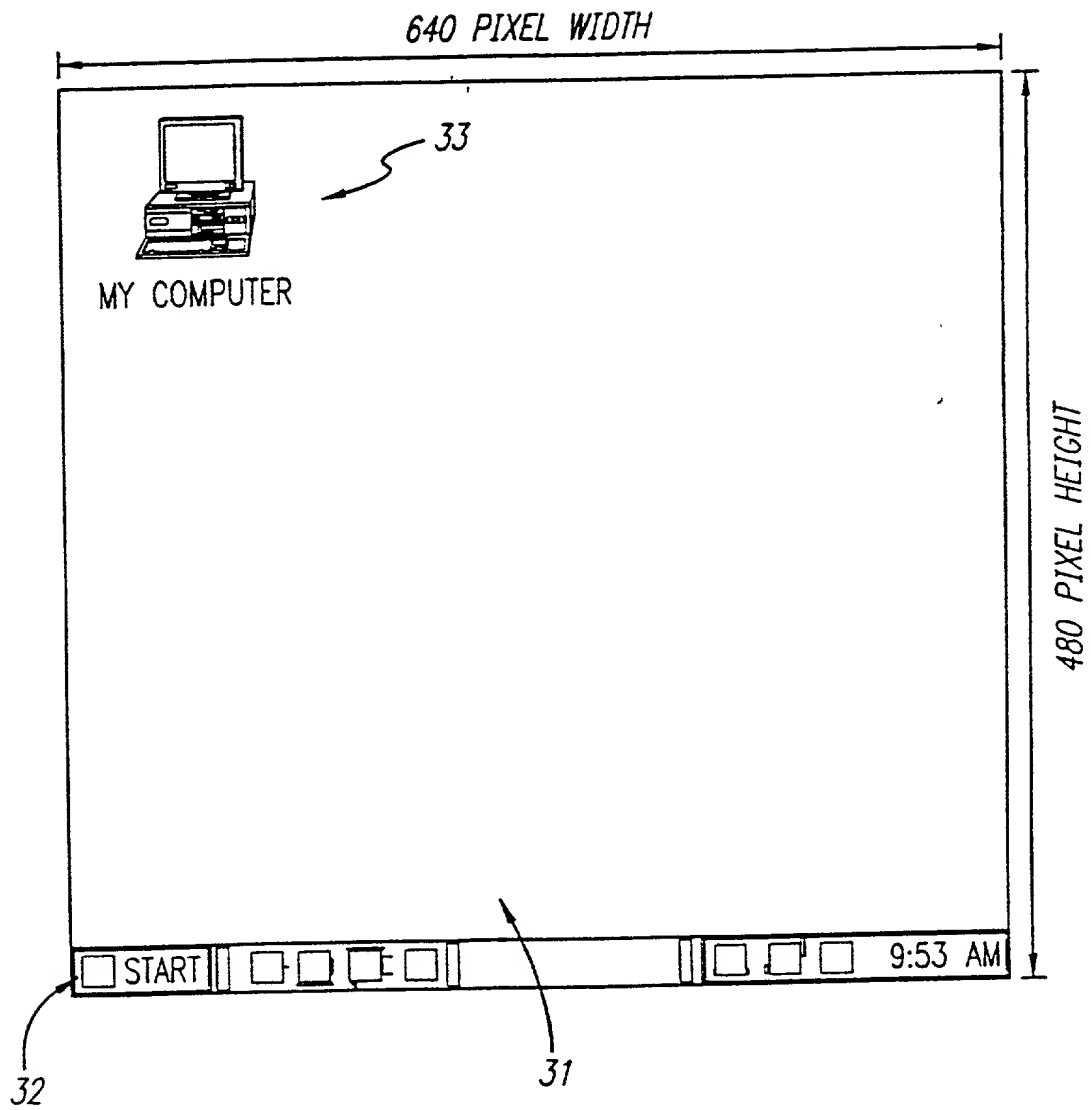
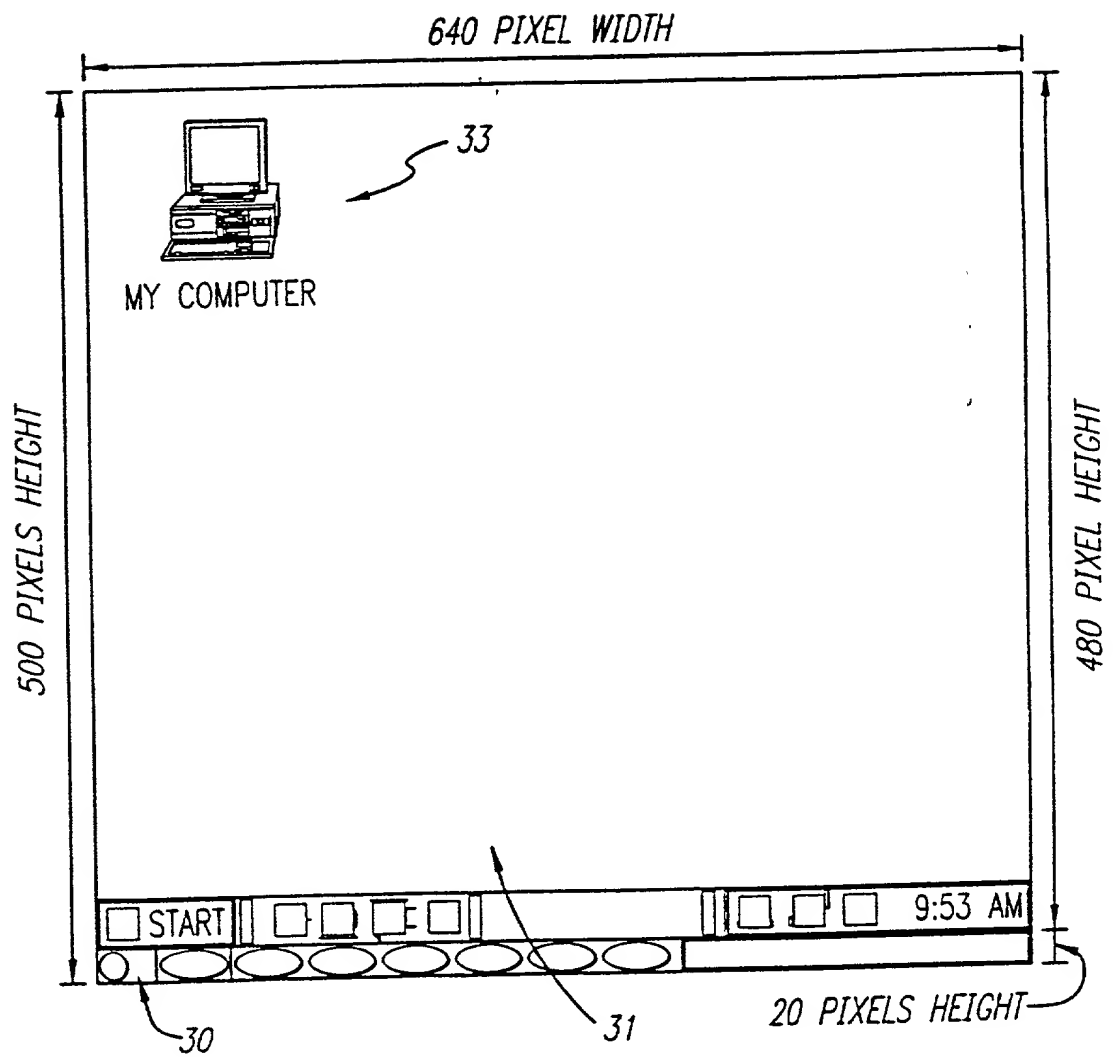


FIG. 16



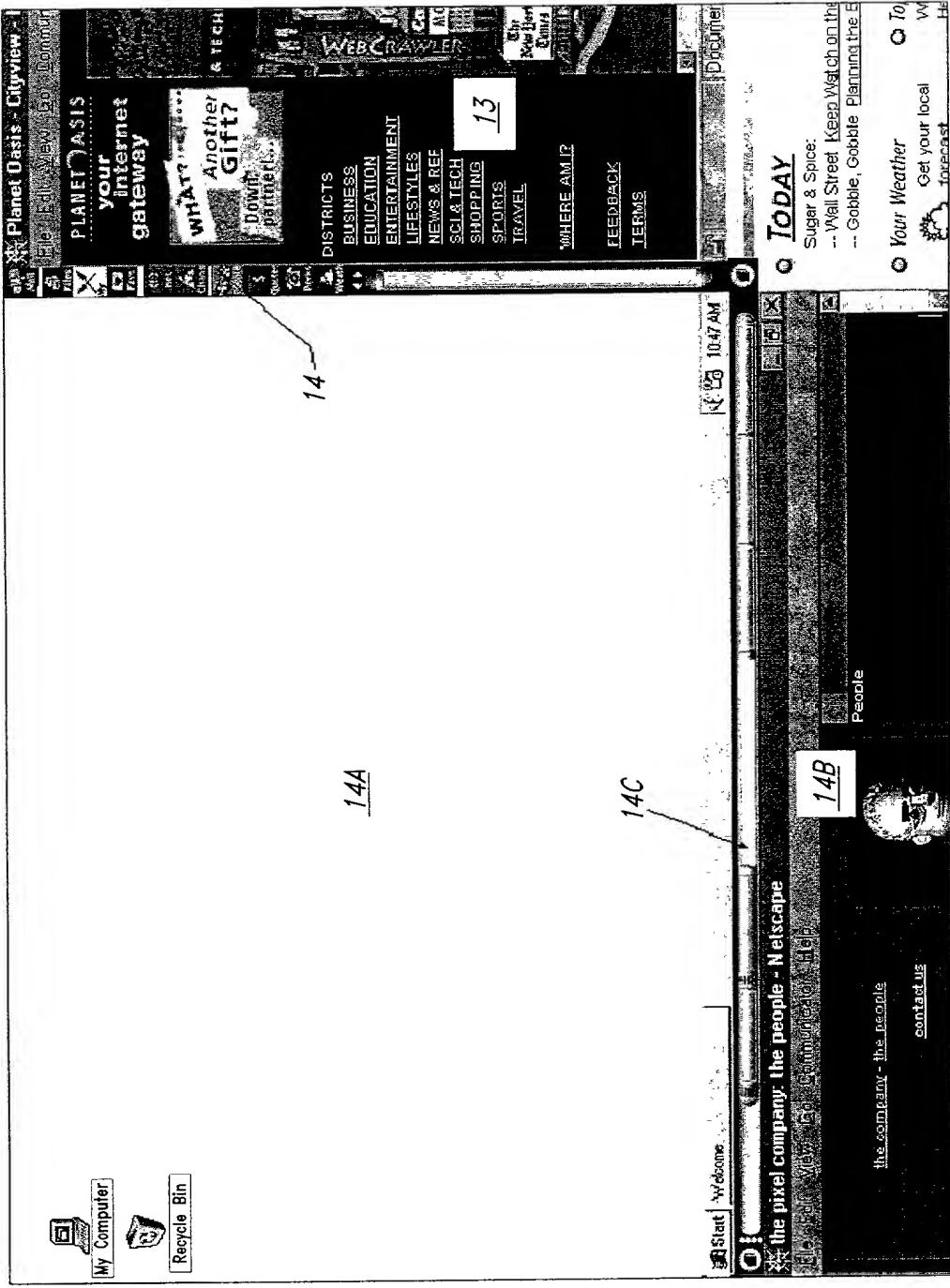


Fig. 17

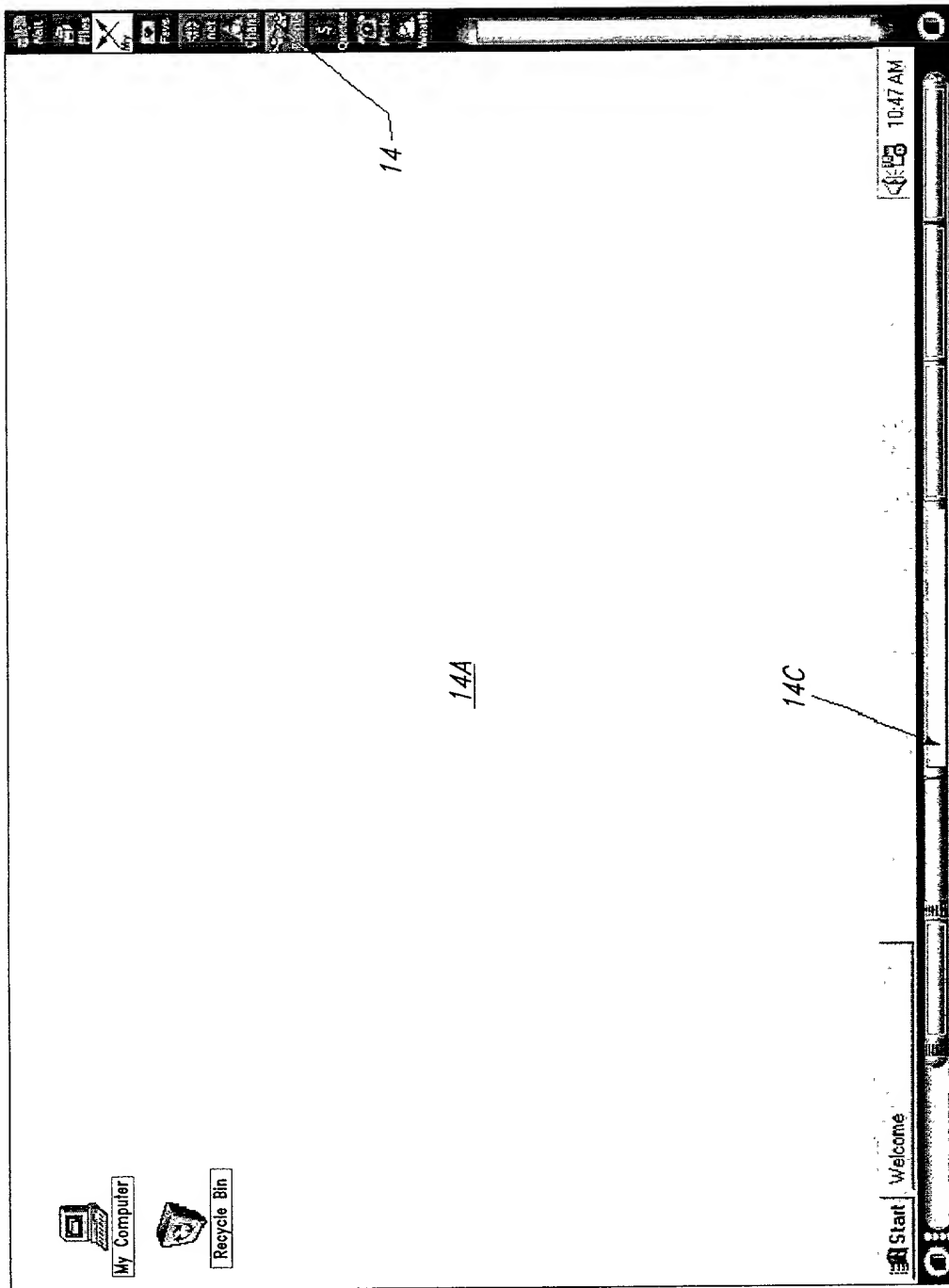


Fig. 18

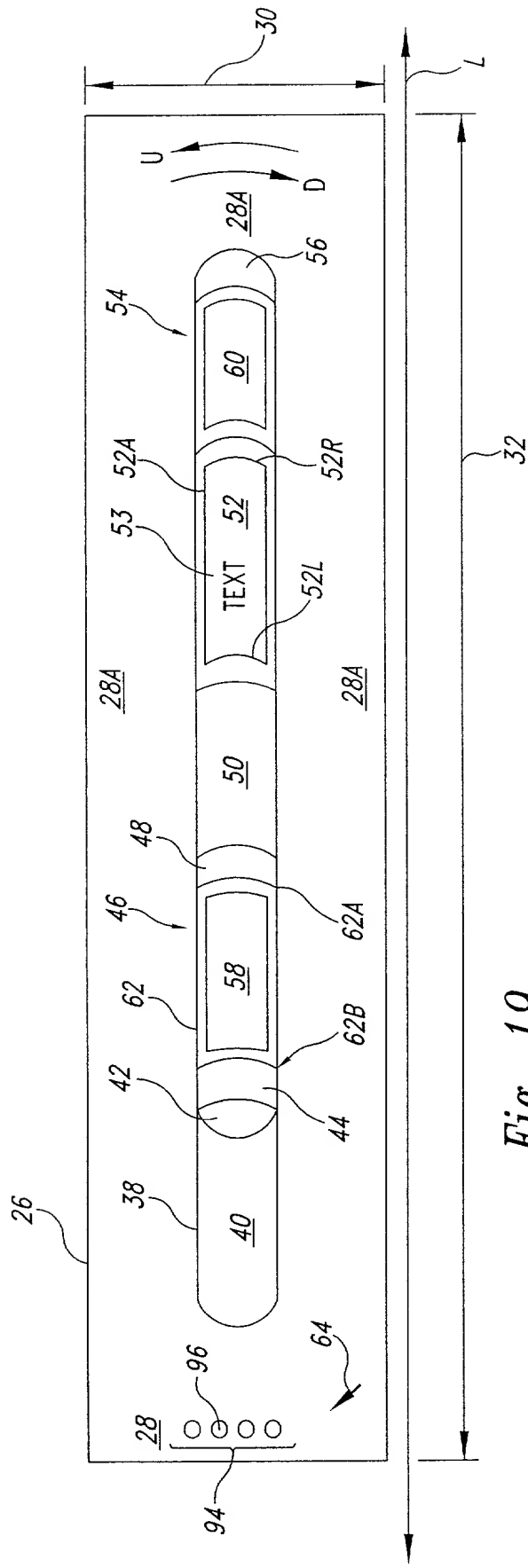


Fig. 19

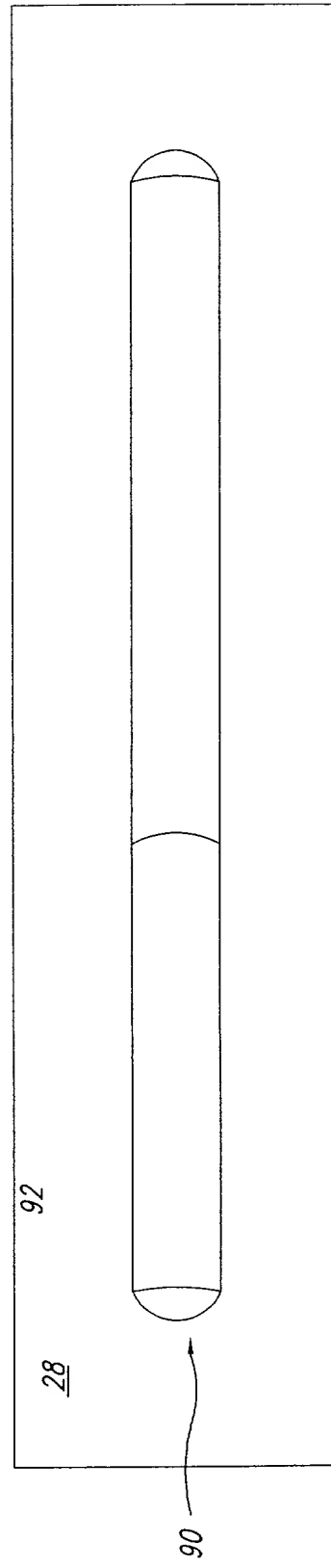


Fig. 21

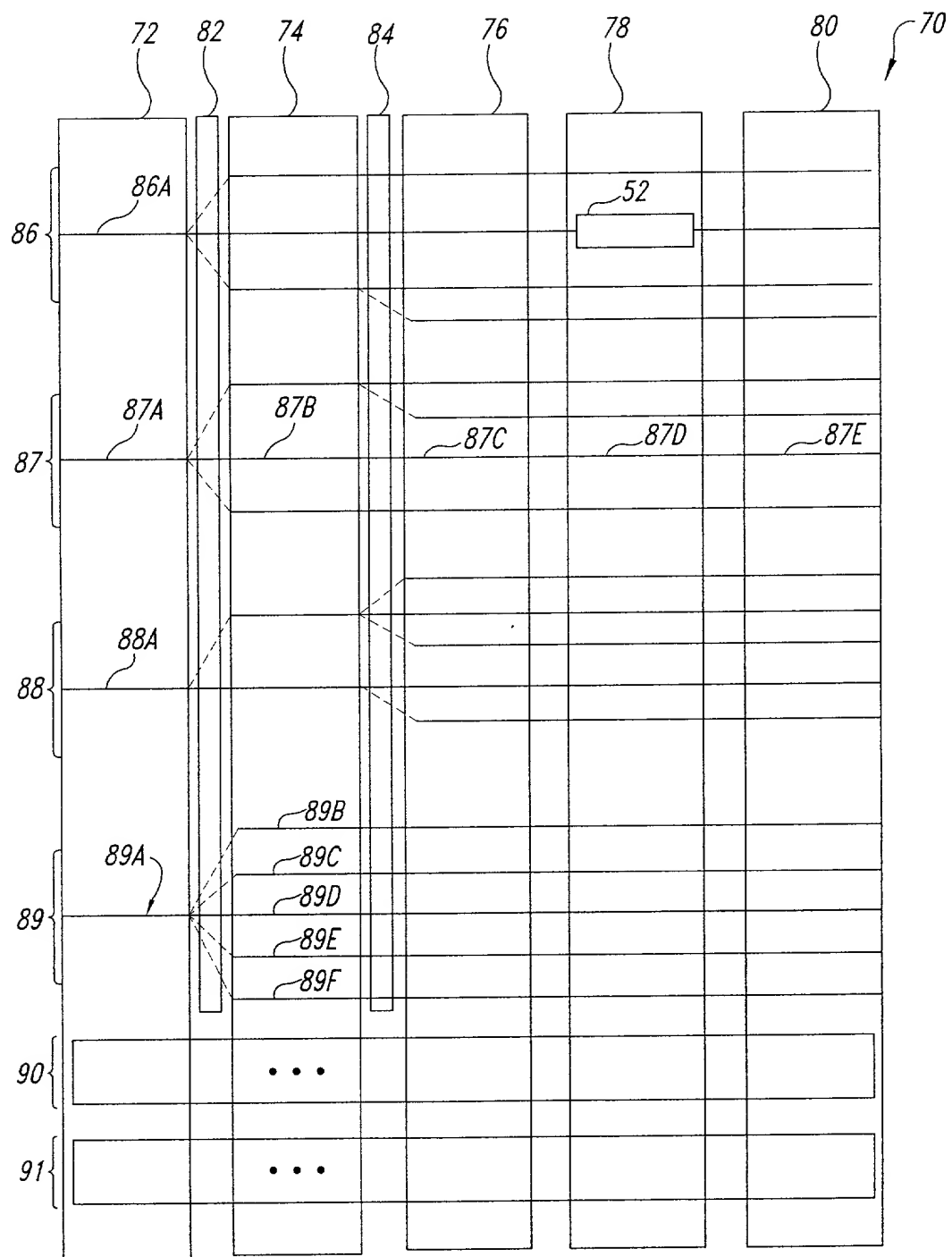


Fig. 20

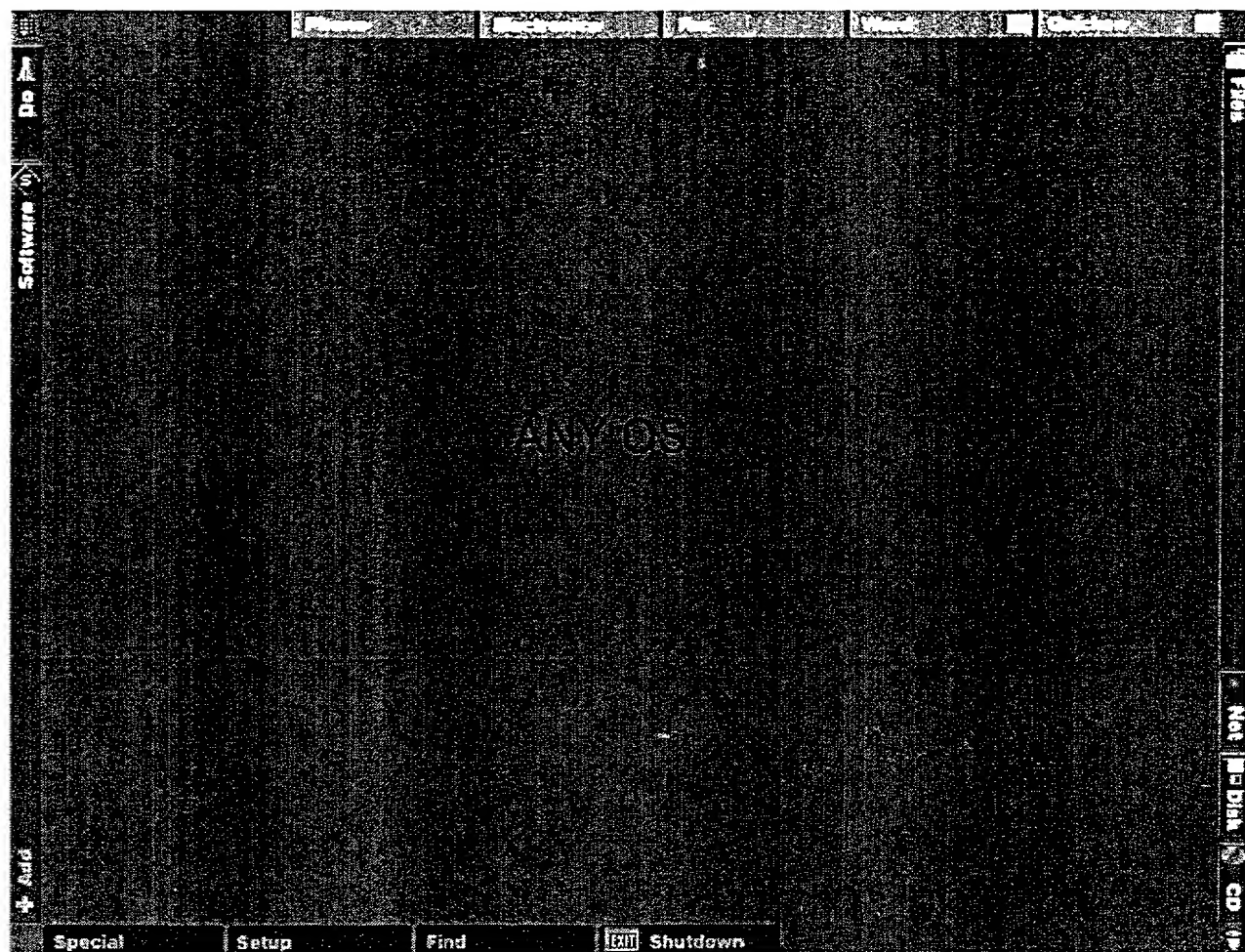


FIG. 22

005664 11:47 AM 11/11/99 11:47 AM

planet oasis | my apartment | stocks | do | favorites | mail | daily | newsstand | searchALL

City View
Business
Education
Entertainment
Lifestyles
News & Ref
Science & Tech
Shopping
Sports
Travel

Arts
Books
Buy Games
Chat Cafe
E-Drive
MovieGuide
Movies
MPlayer
Museums
Music
Play Games
Television

Avery Kids
Bezeik
Castle Thundersteps
Games Domain
Girl Tech
MPlayer.com
Sandbox
SEARCH MORE

NewDownloads
Upgrades
Net Contests
5 Top Games
Cheats
Newsletter
Game FAQs

Start | Welcome | Weather | Business | Sports | Sci-Tech | Entertainment | Travel | In-Depth

10:47 AM

Fig. 23

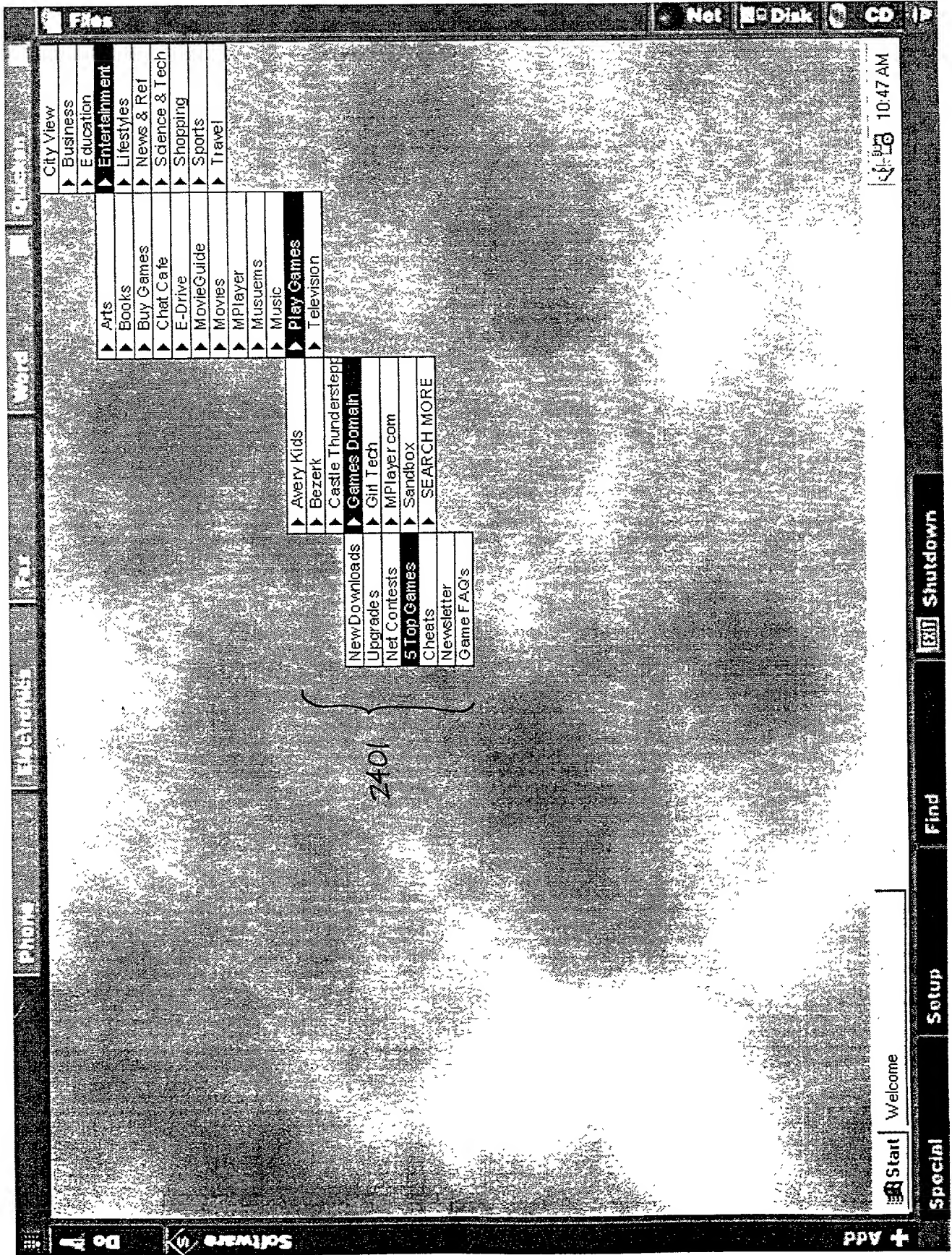


FIG. 24

This is a high-contrast, black and white image, likely a scan of a document page. The image is characterized by a dense, grainy texture. A large, dark, irregular shape dominates the center, resembling a large, dark, textured mass or a large, dark, irregularly shaped object. The background is a lighter, mottled gray, also with a grainy texture. The overall appearance is that of a heavily degraded or high-contrast scan of a physical document.

008277" 3/6/2000

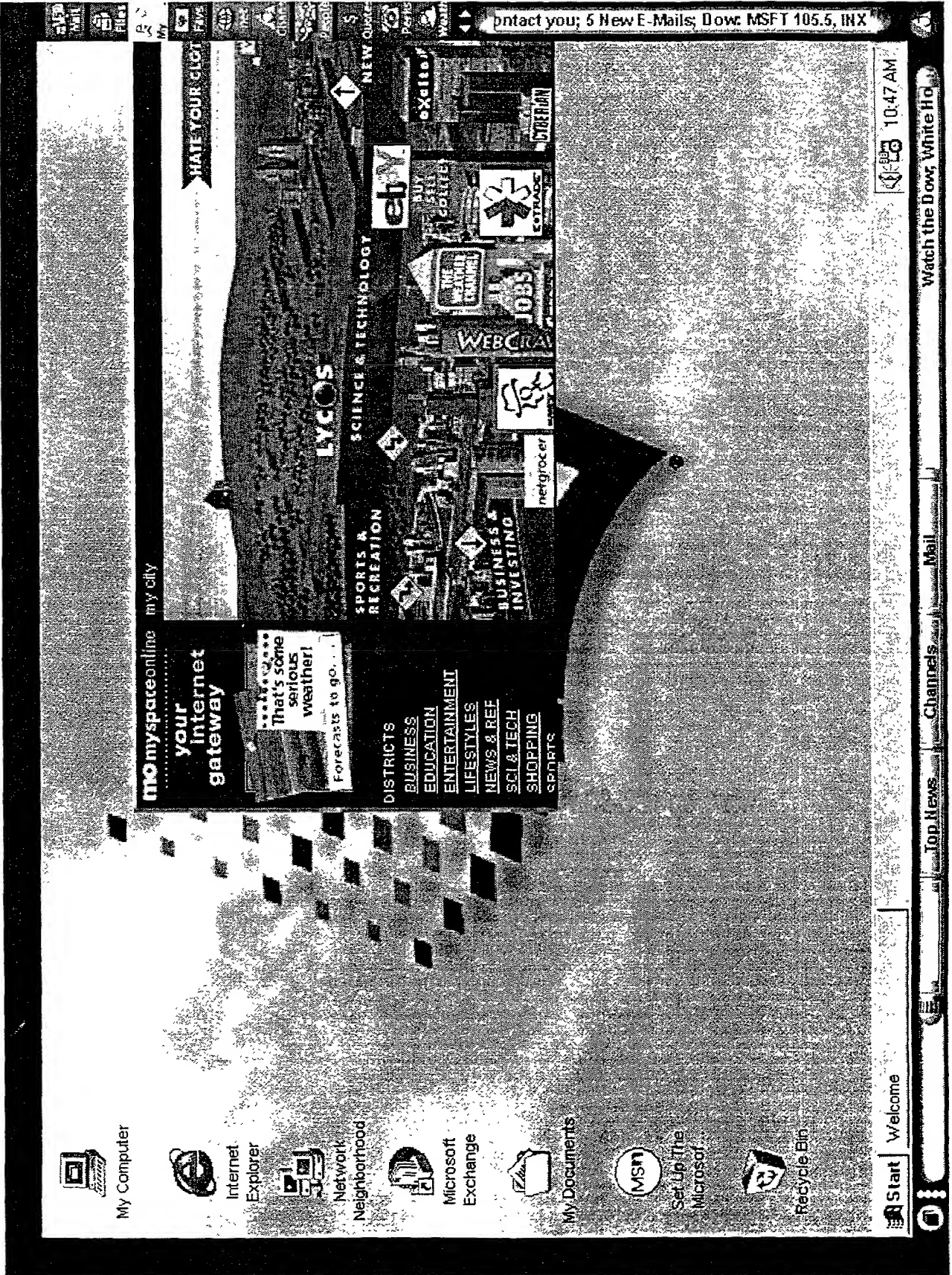


FIG. 26

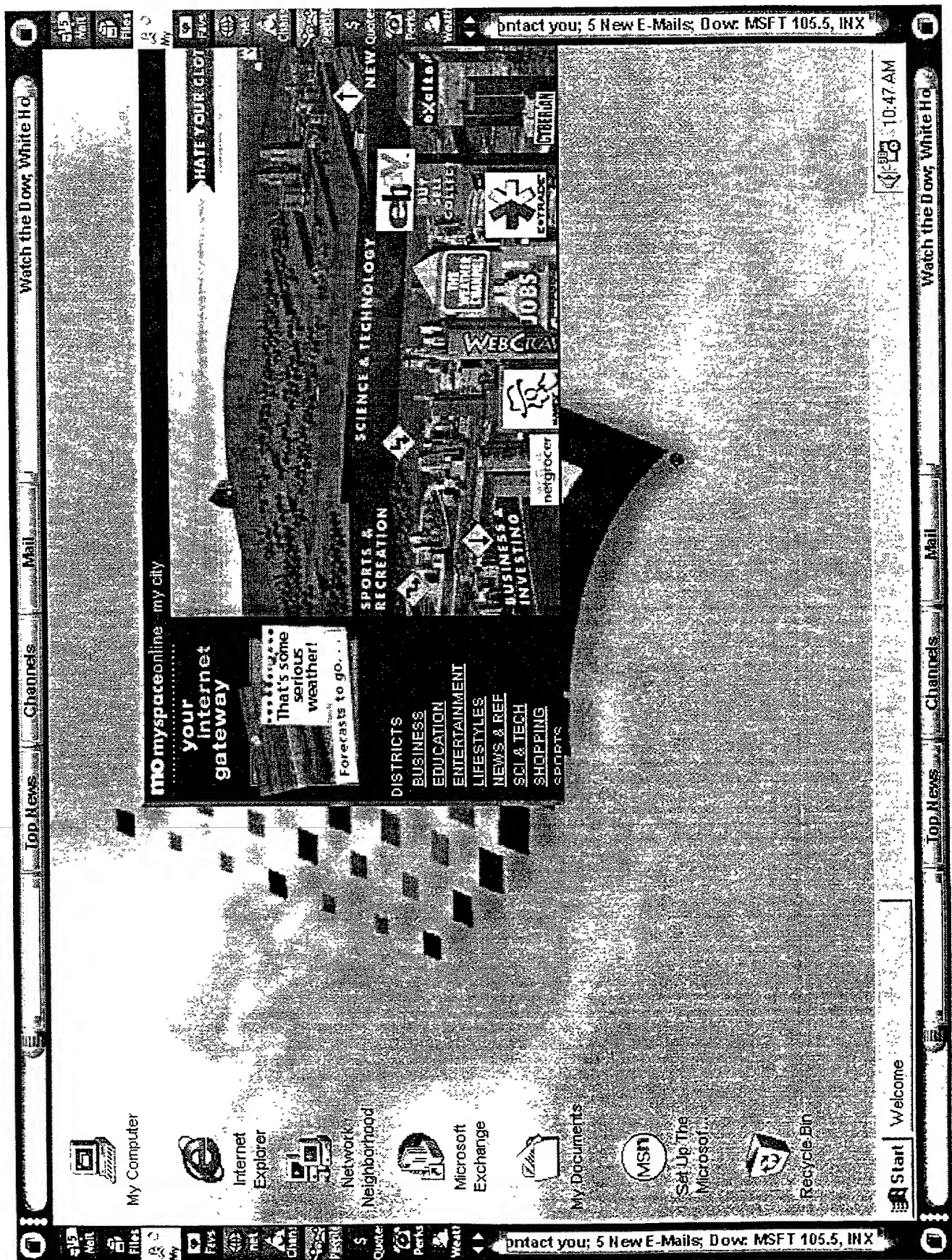
[illegible]

FIG. 27

FIG. 28

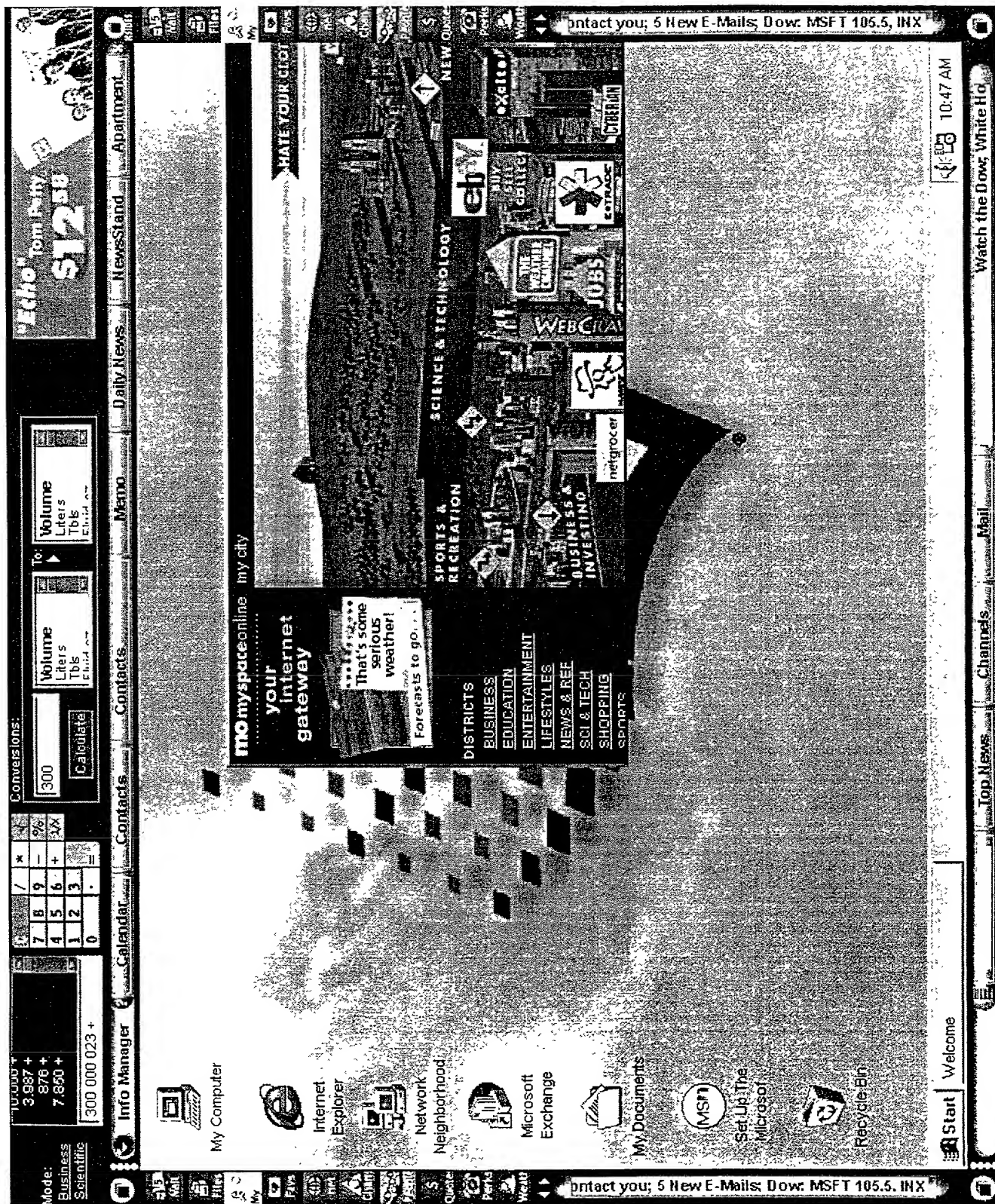


Fig. 29

003211 8/26/2000

Mode:
Business
Scientific

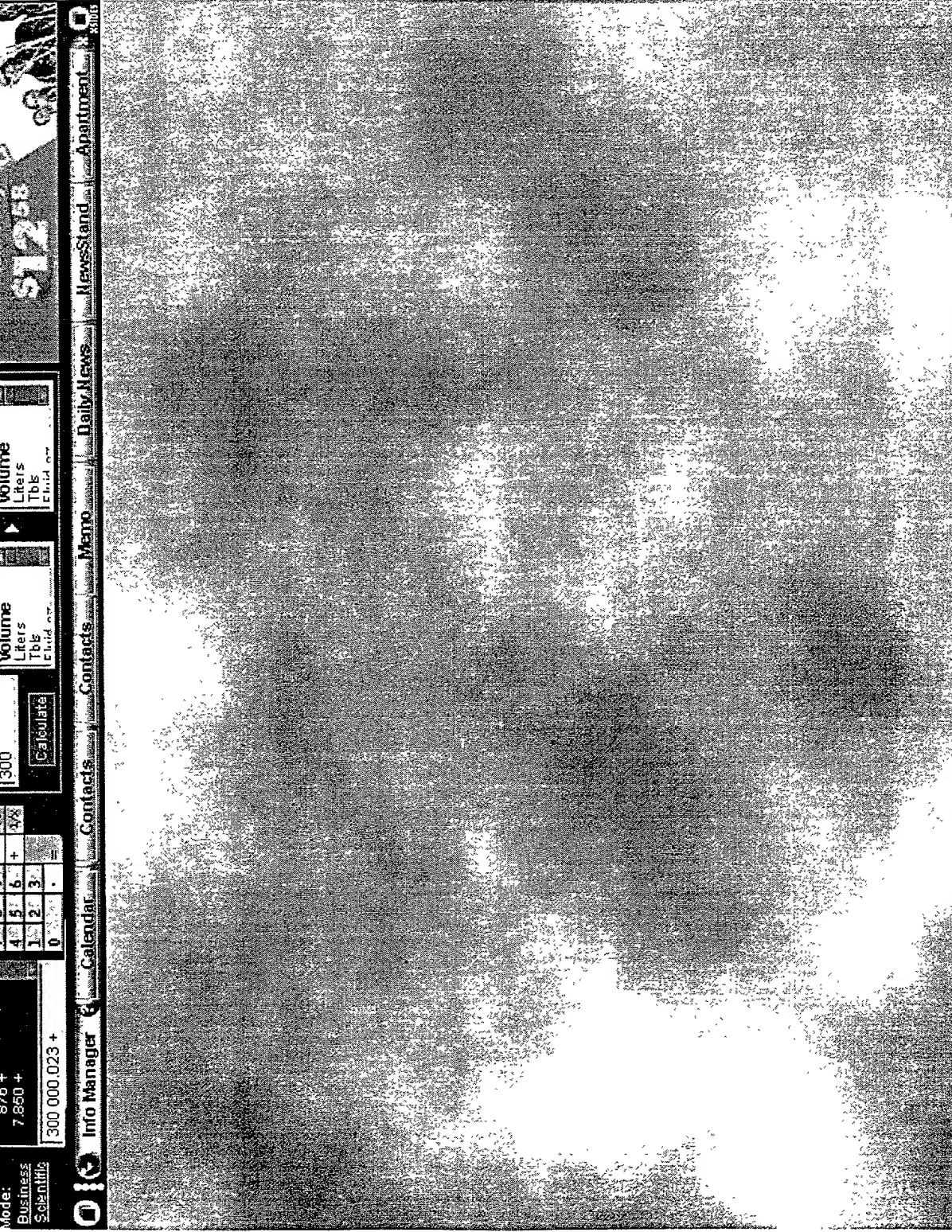
10000 +
3.987 +
876 +
7.850 +
300 000.023 +

0	1	2	3	4	5	6	7	8	9	.	/	*	-	+	=
7	8	9	-	%	1/x										
4	5	6	+												
1	2	3	*												
0	.	*													

CONVERSIONS:
[300] [Volume]
[Liters] [Liters]
[Tbs] [Tbs]
[Fluid oz] [Fluid oz]
[Calculate]

Echo™ Tom Petty
\$12.58

Daily News
NewsStand
Apartment



Start Welcome

10:47 AM

FIG. 30

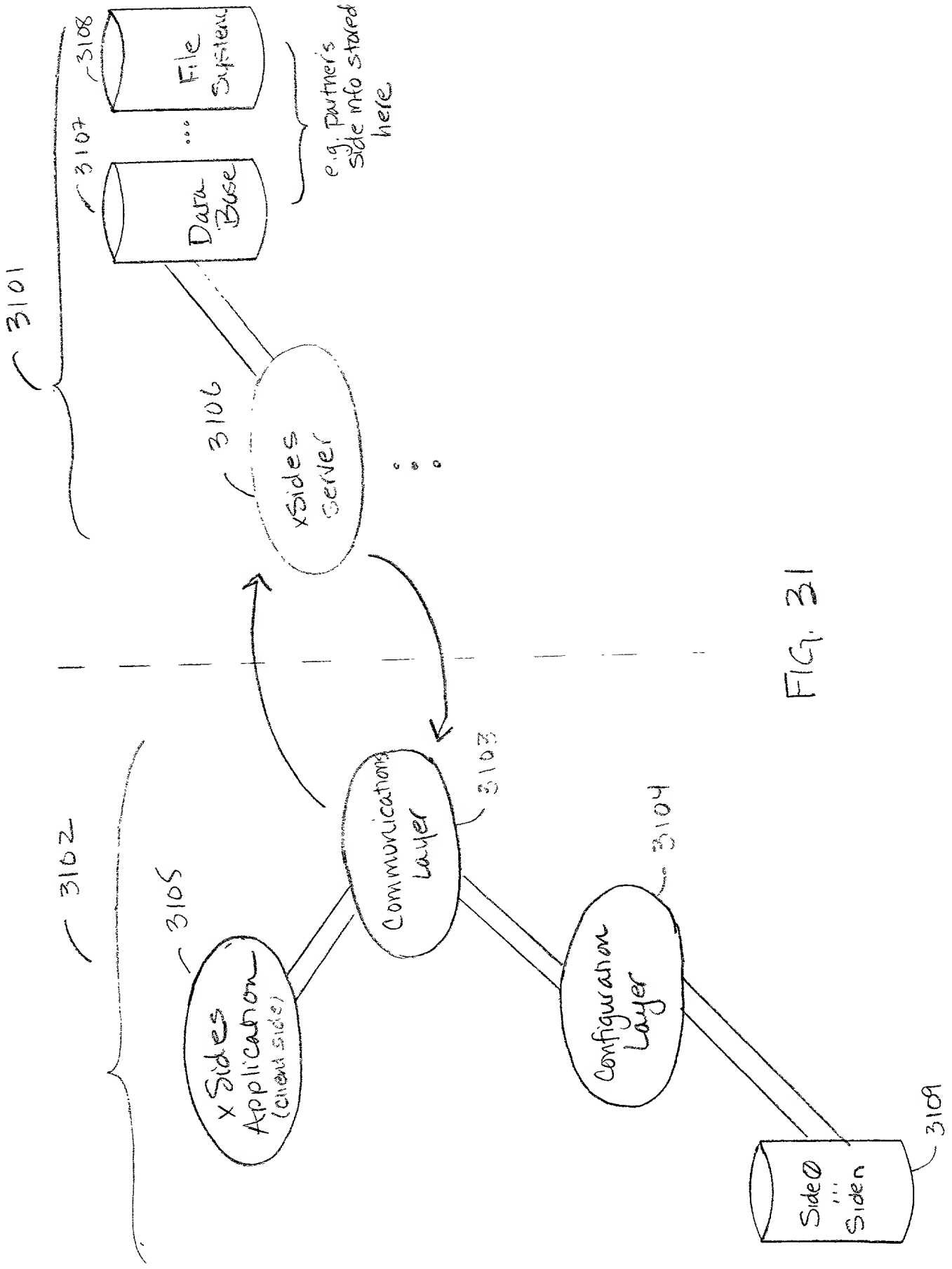


FIG. 31

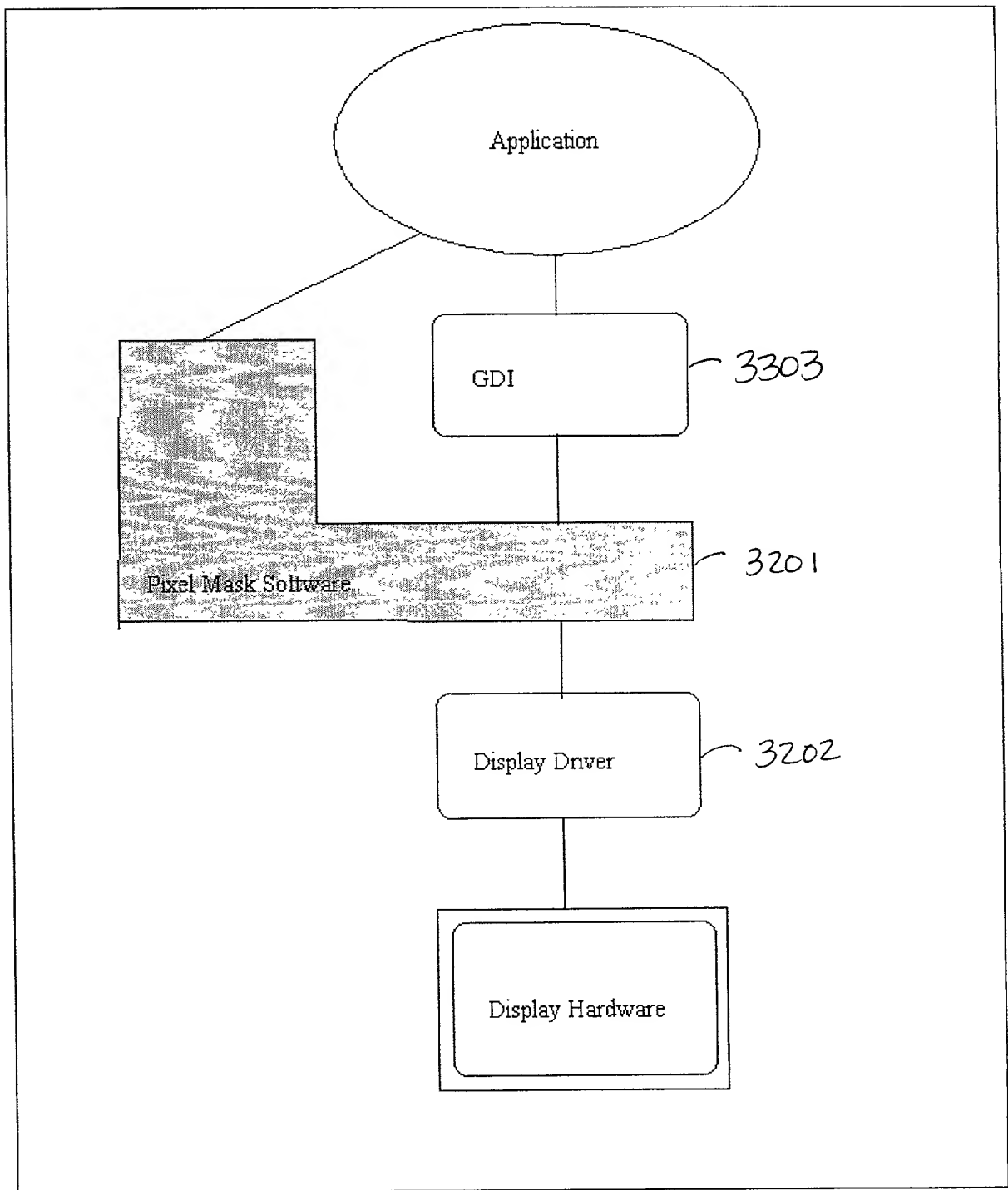


FIG. 32

3304



3301

3305

3302

FIG. 33

Parameter	Value	Unit
Age	1.0	yr
Sex	1.0	male
Weight	1.0	kg
Height	1.0	m
Body mass index	1.0	kg/m ²
Body fat percentage	1.0	%
Heart rate	1.0	beats/min
Stroke volume	1.0	L
Cardiac output	1.0	L/min
Mean arterial pressure	1.0	mmHg
Systemic vascular resistance	1.0	dynes/cm ²
Left ventricular end-diastolic volume	1.0	L
Left ventricular stroke volume	1.0	L
Left ventricular ejection fraction	1.0	%
Left ventricular pressure	1.0	mmHg
Right ventricular end-diastolic volume	1.0	L
Right ventricular stroke volume	1.0	L
Right ventricular ejection fraction	1.0	%
Right ventricular pressure	1.0	mmHg
Pulmonary artery pressure	1.0	mmHg
Pulmonary artery flow	1.0	L/min
Pulmonary artery resistance	1.0	dynes/cm ²
Pulmonary artery compliance	1.0	L/mmHg
Pulmonary artery capacitance	1.0	L
Pulmonary artery volume	1.0	L
Pulmonary artery pressure gradient	1.0	mmHg
Pulmonary artery flow gradient	1.0	L/min
Pulmonary artery resistance gradient	1.0	dynes/cm ²
Pulmonary artery compliance gradient	1.0	L/mmHg
Pulmonary artery capacitance gradient	1.0	L
Pulmonary artery volume gradient	1.0	L
Pulmonary artery pressure gradient	1.0	mmHg
Pulmonary artery flow gradient	1.0	L/min
Pulmonary artery resistance gradient	1.0	dynes/cm ²
Pulmonary artery compliance gradient	1.0	L/mmHg
Pulmonary artery capacitance gradient	1.0	L
Pulmonary artery volume gradient	1.0	L

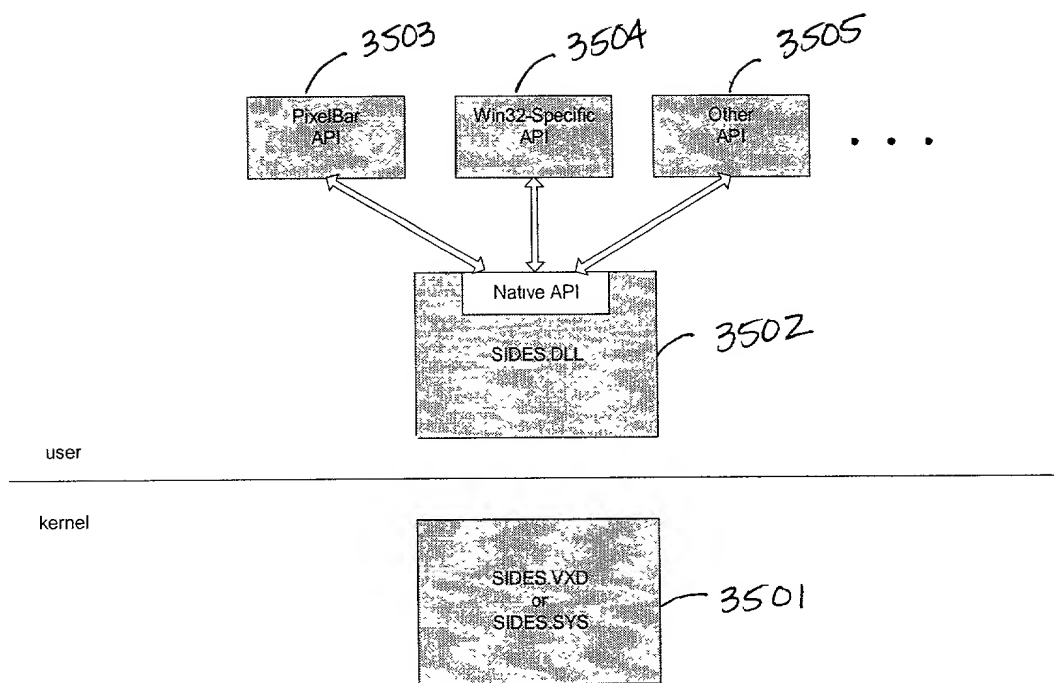


FIG. 35

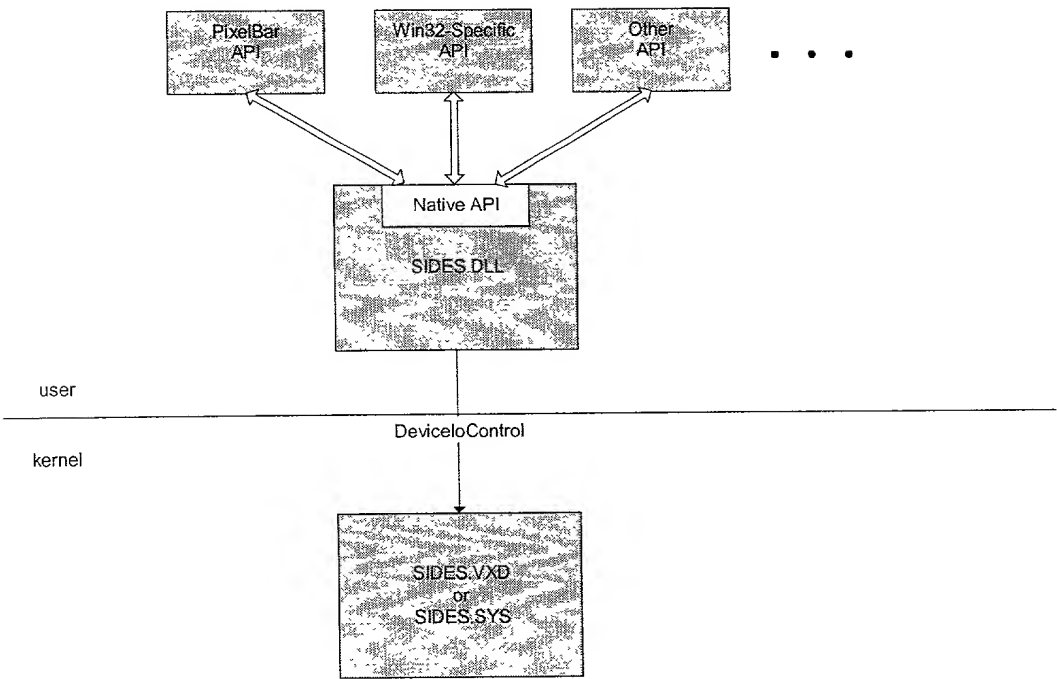


FIG. 36

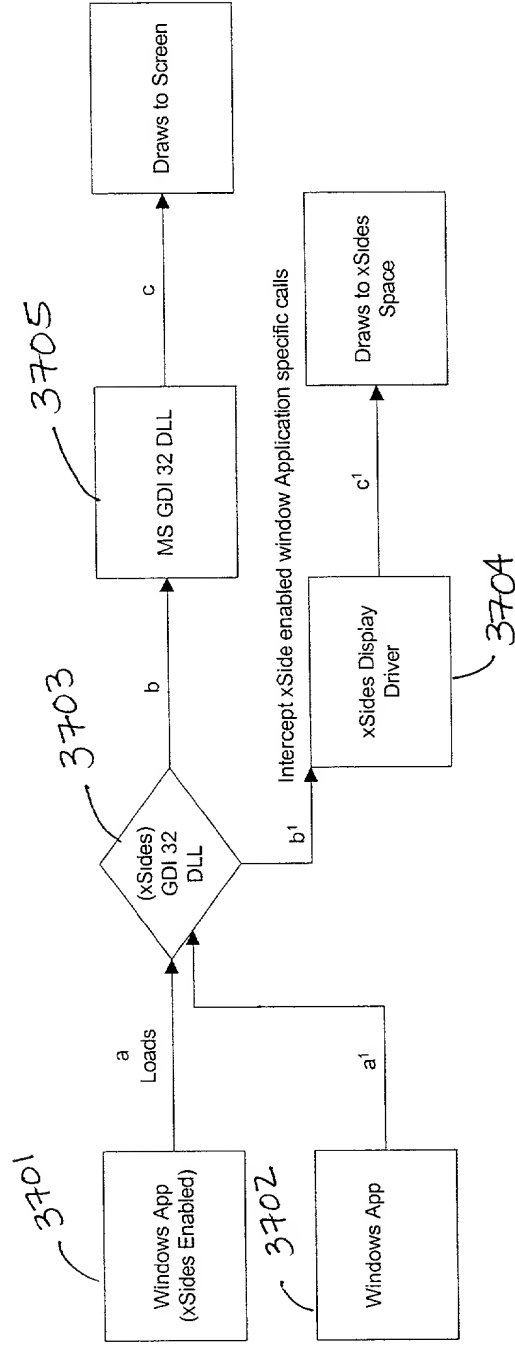


Fig. 37

Appendix A

xSides™ Resides Physically Outside of and Control Independent of Windows

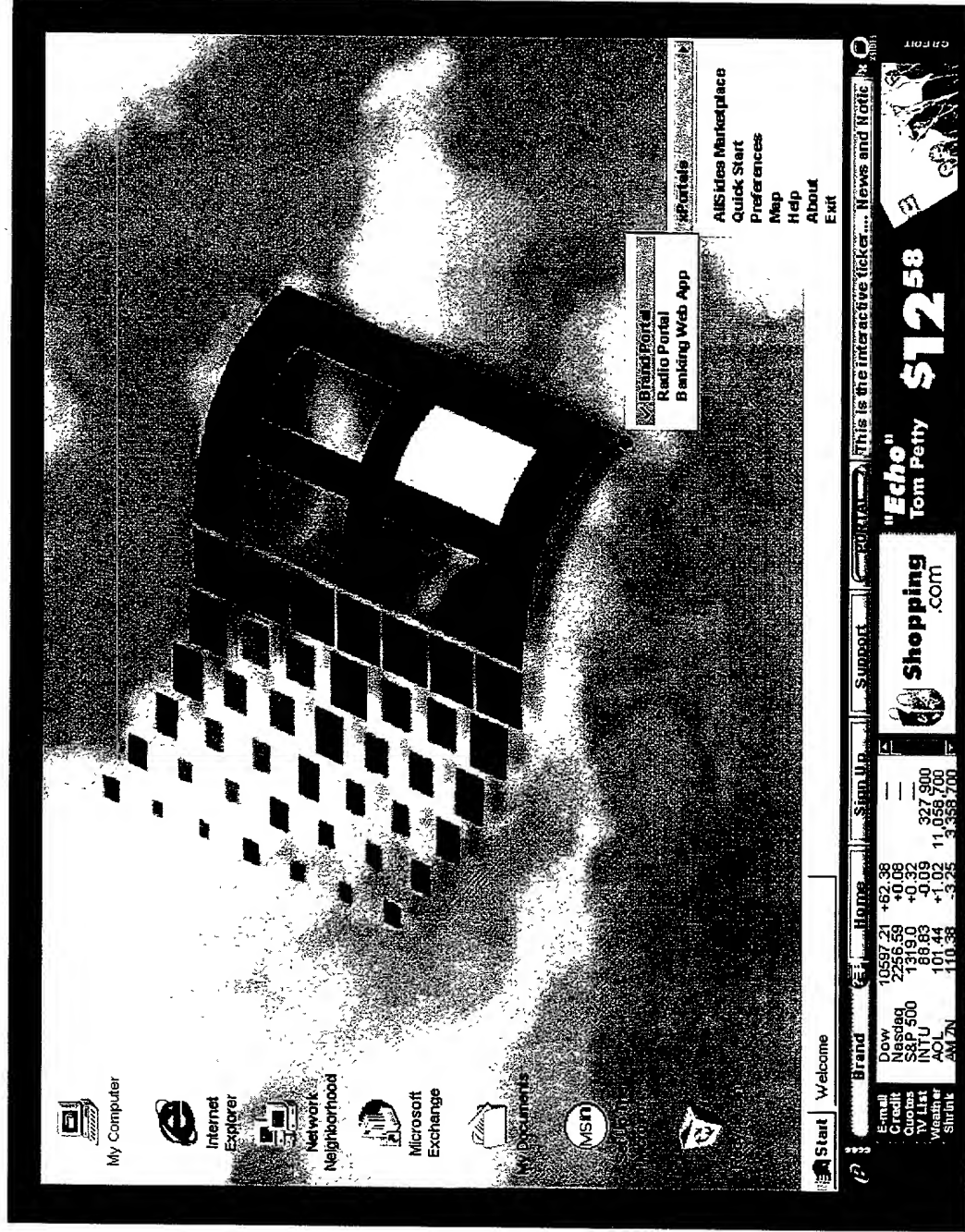


Figure 1: xSides™ resides outside of the computer desktop.

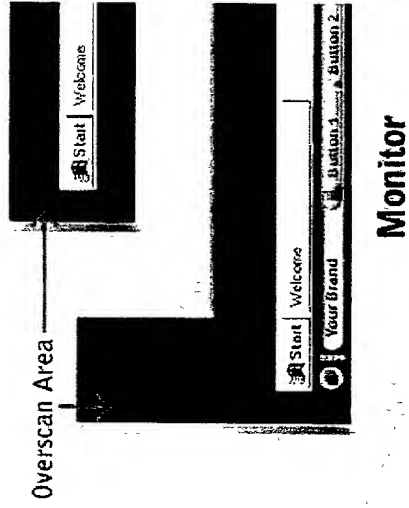


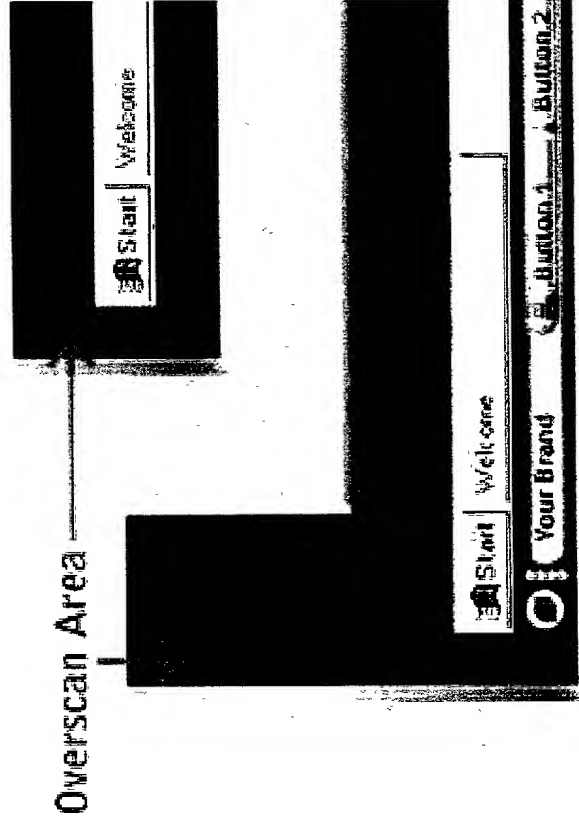
Figure 2: xSides™ initially launches as a cylinder icon with rotating sides.



Figure 3: The “PORTAL” button launches the xSides™ Portal area.



Always Visible and Accessible



Technically Scalable

[illegible]

Figure 4: The xSides™ Portal can contain any html-based image or application.
Below are generic examples of an electronic programming guide and personal information management systems.





Primary Initial Product: xSides™ PortalPack


































2 Primary Applications

- Desktop Mini-Portals
 - We provide the code, they provide the content
 - Best location → increased duration → revenues
 - Partners promote and distribute to their users
- Free ISP Interfaces
 - Persistent display area that is user-friendly
 - With FullView, users can *gain* browsing area
 - Distinct competitive advantage in commodity world

[illegible]

DRAFT "Bates 250

Draft

xSides™ V. 2.2

Product Spec

Including Portal and Web Jump Features

Edited, 8/16/99
12 00pm

xSides V. 2.2 Product Spec

Table of Contents:

Overall Descriptions and Definitions	3
Section 1:	
xSides Portal Design & Feature Guidelines	4
1.1: Descriptions and Definitions	5
1.2: Suggested Space Use Guidelines	6
1.3: Sample Portal Implementations	7
1.4: User Interaction Guidelines	8
Section 2:	
xSides SinglePack Design & Feature Guidelines	10
2.1: The xSides SinglePack Product	11
2.2: Design and Feature Requirements	12
2.3: Function Requirements	14
2.4: Design and Button Function	16
2.5: SinglePack User Cartridge Function	17
2.6: Upgrade and Support Function	18
Section 3	
xSides™ “Web Jump” Design and Feature Guidelines	19
3.1: “Web Jump” Description	20
3.2: Operations	21
3.3: Appearance & Action Requirements, Search Mode	22
3.4: Appearance & action Requirements, General	23
3.5: Web Jump Behavior Guidelines	24

Descriptions and Definitions

The following document outlines the function and design specifications of xSides, a cutting-edge Pixel Company Product .

Our xSides™ Technology enables users and strategic partners to deliver and access content and applications in an area of the computer display which is outside and independent of the computer's operating system. The xSides enabled area cannot be obscured at any time by the Microsoft desktop or any of the applications present therein.

The xSides product initially appears on the bottom of a computer screen as a thin cylinder icon ("the control bar") containing a series of control buttons (see Page 8, Fig. 4). The control bar is comprised of a number of faces, which we call "Sides™," each of which can contain different combinations of content, applications and graphics (hence the name "xSides™").

The user can easily rotate from one Side to the next with mouse clicks to view and access the different content present on a given Side. The minimum number of Sides in each xSides product is three: one for partner content; one for Help, FAQ and Web support; and one for User Buttons that may be modified and manipulated by the user. (See User Cartridge Function, p. 17.)

xSides includes an additional display area called the xSides™ Portal (see Figure 1). This portal expands below the bar and presents an area for companies to display HTML content (email, banner ads, tickers) outside of Windows. There are no design restrictions placed on content that goes into this portal display area.

xSides is designed to deliver a wide selection of consumer products available through a number of distribution sources such as online, PCs, set-top boxes, etc. Therefore, a partner's specific xSides implementation will be able to exist with other such products in this space.

Each partner can create its own "skin." A skin is the packaging of graphical elements that make up a singular visual presentation style of xSides. These elements include a set of buttons (which exists on each xSide face), tickers etc. xSides products allow partners to modify their look and feel by either adopting one of the skins provided by The Pixel Company, or by designing their own.

xSides is available to partners in two versions:

1. **xSides SinglePack:** The SinglePack product includes the xSides Portal feature and allows partners to build an xSides product that modifies functions of the latest Pixel technology, such as site launching, ticker messaging, and Help functions. Partners can modify the look and feel of xSides within defined parameters (outlined by The Pixel Company), or they can work directly with The Pixel Company to design new, partner-specific design guidelines. Function, size, and number of cartridges will be restricted.
2. **xSides PrePack:** PrePack contains The Pixel Company's pre-packaged partners as content on the bar. The PrePack version will have limited functionality and modification abilities.

Section 1

xSides Portal Design & Feature Guidelines

1.1: The xSides Portal: Descriptions and Definitions

The xSides technology includes an expanded area where partners can display HTML content (tickers, links, banner ads). The xSides Portal is available to strategic and content partners who wish to provide products and services to their users in an area that is always available and visible--without getting in the way of computer and Web functionality.

The Portal area functions exclusively with the xSides control bar (see Section 2, p. 10). The xSides Portal launches underneath the bar, and responds to xSides commands, such as open, close, refresh, etc. xSides itself functions as stated in Section 2 (p.10) of this document.

The Portal area will always be 70 pixels high x the horizontal resolution of the screen, minus 20 pixels on the right hand side of the Portal area reserved for user controls.

There are no restrictions on design look and feel for content presented in the xSides portal display area. Content partners may work with The Pixel Company to create their own unique design. There are space-use guidelines (p.6) to follow.



Figure 1

1.2: xSides Portal Suggested Space-Use Guidelines

Design specifications are kept open to respond to the requirements of each Partner. However, for product design consistency, the following guidelines should be considered when standard xSides Portal functionality is presented.

1. **Background**
The xSides bar should sit on a black background of the overscan area in which it resides.
2. **Constant Distributor logo**
The left-hand side of the xSides bar will display the Distributor's logo at all times and will remain visible throughout all rotations and uses of each of the bar's faces.
3. **Portal Content logo**
This logo will remain until the user displays another xSides Portal.
4. **Application or link areas**
The xSides Portal gives users access to Web applications, sites and resource links, and tickers. Tickers display scrolling text that, when clicked on, can launch an associated Web page. Text and hyperlinks can be updated as defined in each Partner's contract.
5. **Banner display area**
Several banner sizes are supported in this areas if the Partner chooses to include them.

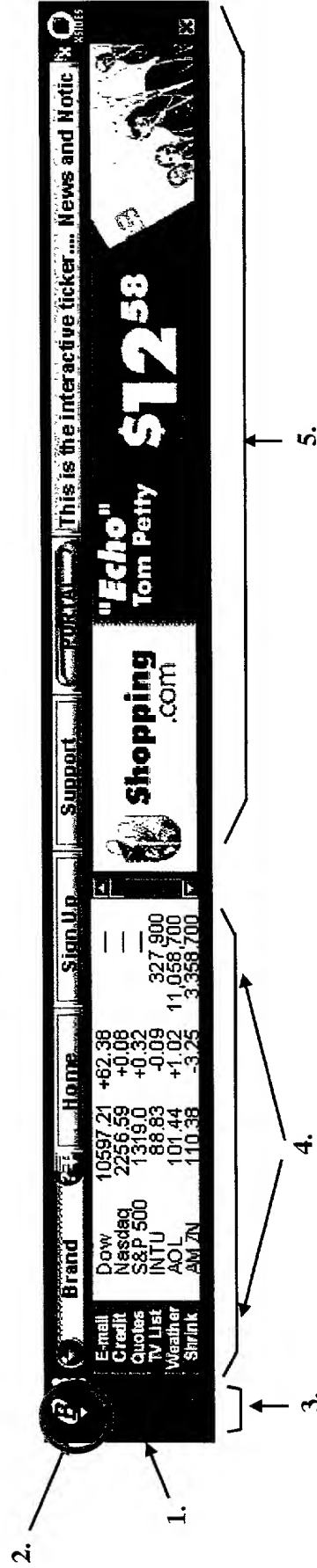


Figure 2

1.3: Sample xSides Portal Implementations

Unlimited new products may also exist in the xSides Portal area. These may include a Calendar and Personal Information Manager (PIM), Stock tickers, etc.. Here are just a few more examples of different Portal implementations.

1.4: xSides Portal: User Interaction Guidelines

Minimizing the xSides Portal

The first time a user opens an xSides product which includes the Portal area, it can launch into its maximized view. This open option may be turned off by the user as well. Each xSides control bar with the Portal feature will have a prominent "Portal" button on the Side. Once the product has been installed, the user clicks this button to show, refresh or hide the xSides Portal area. Each Portal area also has a familiar "X" box in the upper right-hand corner. Clicking here will either launch the Portal, or make it retreat to show simply the desktop, and the xSides bar.



Figure 3

Maximizing xSides Portal Again

When the xSides Portal is in its minimized state (i.e., turned off), the user clicks on the Portal button to bring xSides Portal back to its full viewing size.



Figure 4

1.4: xSides Portal: User Interaction Guidelines, continued

Quick-Jump Feature Between Portals

Users may install and merge several xSides products with Portals. Once this is done, users will be able to quickly jump between them by using a cascading menu system accessible through the xSides logo on the right hand side of the bar.

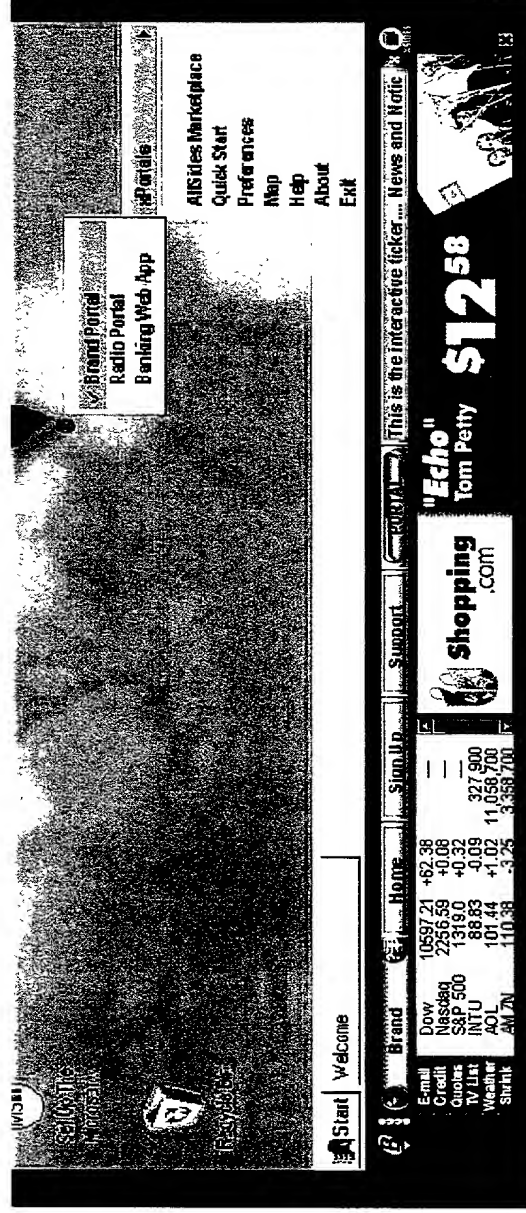


Figure 5

Section 2

xSides SinglePack Design & Feature Guidelines

2.1: The xSides SinglePack Product

xSides Single Pack

xSides provides Partners the option of changing the look and feel of xSides within The Pixel Company's guidelines. A specific set of buttons, or "skins," give Partners a space to customize the way they appear on the xSides bar. Partners may also work with Pixel to have a unique skin built for them, on a for-hire basis, and time/resource permitting. The following restrictions apply to xSides presentation and function:

- The size of the control bar will be the same for all xSides products. This will enable users to mix-and-match xSides bars as they wish, and as they discover new xSides applications in the marketplace.
- Each xSides product will include two additional Sides--a "User" Side and a Help/Support Side. (See "User Cartridge" and "Upgrade and Support" sections, p. 17, 18.)
- The user will be able to remove this application from the computer screen by exiting and/or uninstalling the program from their system, either as part of a larger group of xSides cartridges, or as a stand-alone product.
- xSides products have a limited set of buttons and functionality that must be present and supported in every instance of the product, in addition to Partner-defined buttons.
- The xSide product must be able to rotate.
- The xSide product must contain logos.

2.2: xSides Design and Feature Requirements

1. Preset Buttons

The preset function allows users to "bookmark" specific Sides. Users can set a their selection (CD Player, News & Sports, etc) by right-clicking on the preset button when that particular Side is visible. Once this has been done, users can return to their pre-set destination by left-clicking on that button. Preset Buttons change color once a setting is saved.

2. Category

The major category titles are displayed here, and clicking on then switches categories.

3. Help/?

When the user clicks on the Help/? button, a pop-up window is launched that contains Welcome, Map, Preferences, Help, and About xSides information.

4. Rotator turns

When clicked, Rotators turn the portion of the bar to the right of the Rotator; left click turns to the next Side, right click back.

5. Pixel logo

When cursing over over the Pixel Company/xSides logo, this animated graphic will rotate. The logo button, when clicked, links to The Pixel Company Web site.

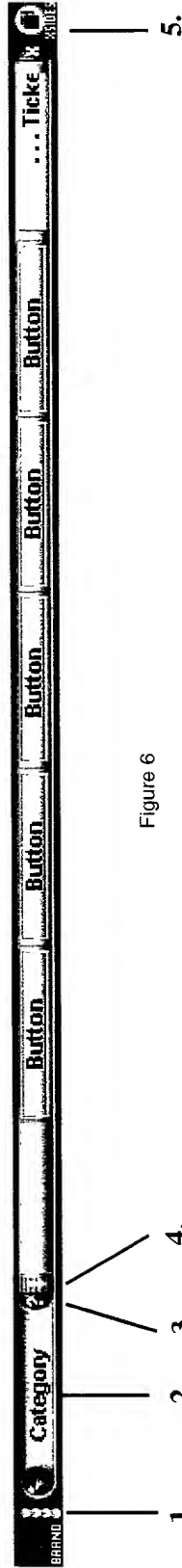


Figure 6

Please Note: The xSides PrePack design will use the standard xSides look, as shown in the example above--with the same feature functionality as the xSides Single Pack.

Figure 7

2.2: xSides Design and Feature Requirements, Cont.

6. **Partner Logo**
Partner logo is displayed here. A left mouse click links to Partner's Web page.
7. **Non-Action buttons**
Non-Action buttons function only as title buttons or place holders. They don't launch or highlight and cannot be depressed with a mouse down or keyboard command.
8. **Action buttons**
Action buttons are depressible and launch Web links. They highlight and function as described under Button Highlighting. (See "xSides Functions" section).
9. **Ticker**
Tickers display scrolling text and, when clicked on, can link to an connecting Web page. Text and hyperlinks can be updated as defined in each partner's contract.
10. **Exit / (X)**
When the user clicks on the Exit button (X), a window appears that confirms the user's choice to exit xSides.

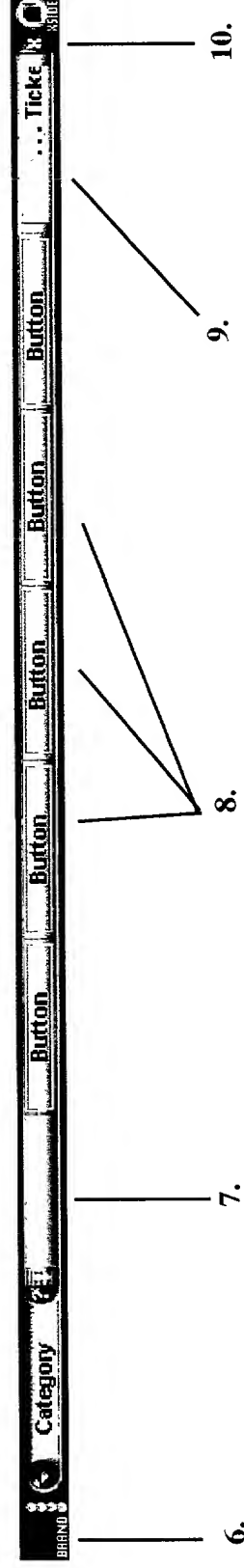


Figure 8

Note:

Logo and Branding Guidelines: The xSides logo and branding is incorporated into the design of the product and the product's download button(s). The xSides/Pixel logo and branding signify that a company uses The Pixel Company's patented technology. The logo and branding can be used to enhance partner marketing materials, including brochures, presentations, and advertisements. The Pixel Co. and xSides logo and branding must be used according to the guidelines outlined in the deliverables. Any deviation from these guidelines must be approved by The Pixel Company.

2.3: xSides Function Requirements

Startup Sequence:

Once the user has downloaded, installed, and launched the xSides program, the user is prompted by a dialog box to decide how they want the product to load. The default here is for automatic load during the Windows boot sequence. While the product software loads, a text message is displayed in the horizontal blank area just below the Task Bar, informing the user that the program is "Adding Overscan Area to Display" or "Initializing xSides Display." (xSides startup does not interfere with the display of startup sequence of Windows.)

At Startup:

Following one of the above text messages, the xSides bar appears, and the Distributor or Sub-Distributor cartridge is displayed--unless the user has set a different startup category under Preferences, accessed through the red "?" button. The first time xSides runs, the Welcome screen appears when the cursor is moved anywhere over the xSides bar.

Rotation Action:

The Rotator turns different elements of the bar. A left-click turns the bar upward to the next face, and a right-click turns it downward to the previous one. The Rotator is the strip located on the left side of the bar, to the right of the red Help "?." The buttons to the right of the Rotator is the section that turns and changes.

Action Buttons:

Action Buttons, when clicked, launch an associated function--application, URL, document, CD player, etc. Buttons with no action function as title areas only, have no associated hyperlink, and do not highlight.

Ticker

Tickers display scrolling text and, when clicked on, can activate a hyperlink to launch an associated Web page. Text and hyperlinks for tickers can be changed and updated. Users will receive the updated Ticker text and URL when they connect to the Web, and xSides is running on their machines.

Button Highlighting:

Mouse over: At "mouse over," a lighting effect appears in the lower-left corner of the button frame.

Mouse down: When the left mouse button is depressed, highlighting indicates that the xSides button is depressed.

Mouse up: When the left mouse is released, the function is activated, and the xSides button appears to spring back, reverting to "mouse over" state, described above.

2.3: xSides Function Requirements (cont.)

Sound:

A subtle sound effect accompanies the rotating action of the bar , to enhance the xSides experience.

Pop-Up Window:

A pop-up window will launch when the user clicks on the "?" button. The pop-up will have tabs such as Welcome, Help, About, Map, and Preferences.

Screen Saver Mode:

The xSides bar slowly rotates through all content when the computer is in Screen Saver mode.

Pop-Up Help

When the cursor rests over a button, a pop-up window appears. Pop-up Help contain sub-level menus, descriptive text, and more.

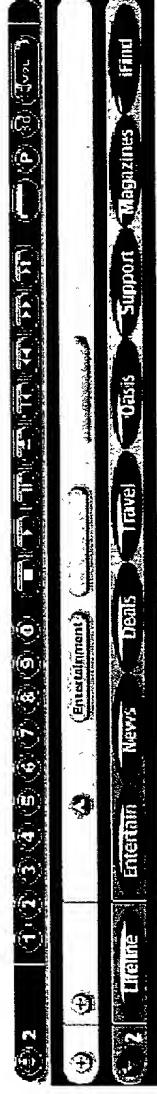
Upgrade Path and Merging:

The xSides upgrade/merge functionality provides some control and flexibility to the user, keeping the application fresh, useful and personalized, one side at a time. xSides technology is designed to be delivered a la carte to users, with the advanced ability to allow users to add new product features and merge any xSides branded products they choose to any other previously installed xSides product(s). Users will be able to upgrade and merge their xSides products via download from our partners', or an xSides web site (under construction). The user will have full discretion to download any number of xSides products.

The Distribution partner maintains persistent branding and content throughout this merging situation. The Distributor's logo and content remains in their current location no matter what other cartridges have been added or deleted. To remove the distributor branding and content, the user must completely uninstall the product.

2.4: xSides Design Themes

xSides provides Partners the option of changing the look and feel of xSides within The Pixel Company's guidelines. A specific set of buttons, or "themes," gives Partners a space to customize the way they appear on the xSides bar. Partners may also work with Pixel to have a unique theme built for them, on a for-hire basis, and time/resource permitting.



(Example of different product "Themes")

Figure 9

2.5: xSides User Cartridge Function

Description: User buttons are found in the User cartridge in xSides. This Cartridge allow users to customize buttons to suit their needs. xSides products come with seven User buttons

Buttons can be added to User Cartridges in a number of ways. Users may add links thereby creating buttons by dragging and dropping (or copying and pasting) into user button frame.

Additional bonus User bars can be added to the User Cartridge when installing another Partner cartridge (each partner cartridge contains a User section. The new User bar is added, titled with the partner's name, containing the link that was just selected.

To clear a User button, right-click on the button, then left-click on "Delete" in the pop-up menu that appears; or, select and highlight a User button and press the DEL key on your keyboard.

To rename a User button, right-click on that button, then left-click on "Rename" in the pop-up menu that appears. Type the new name in the field that appears, then hit "Enter" on your keyboard or click on "Yes" in the pop-up menu. To cancel the change, hit "No."



Figure 10

2.6: xSides Upgrade and Support Function

Description: Each xSides product has a cartridge specifically designated for support and upgrade use. Using these cartridges the user can access links to frequently asked questions, upgrade pages where other xSides products can be found, and receive news and information about xSides products.

This cartridge has one rotation containing a combination of button links and tickers.



Figure 11

2.7: Screen Saver Function

Description: All xSides products will remain on screen at all times, even when the screen saver is activated. The xSides bar and portal will function as specified, slowly rotating through all content.

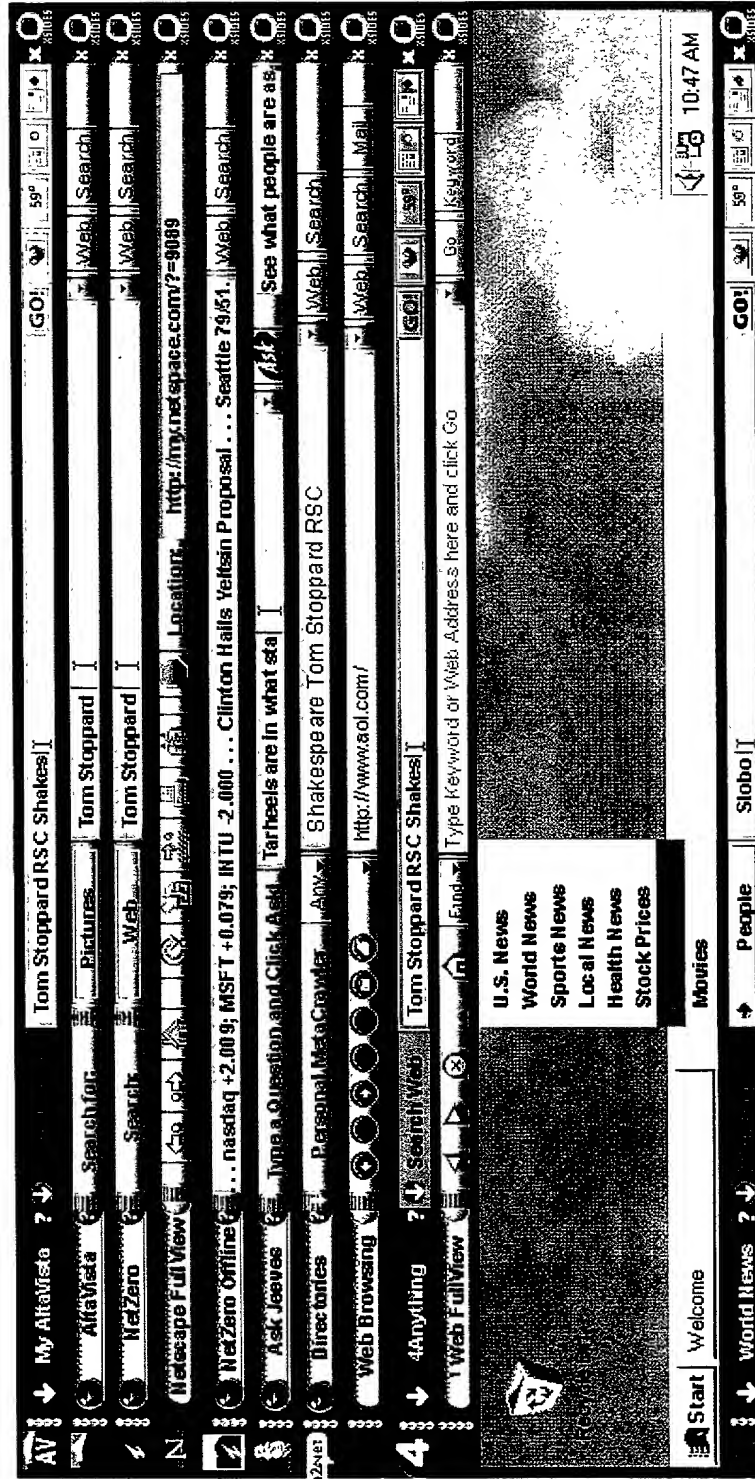
Section 3

xSides Web Jump Design and Feature Guidelines



3.1: xSides “Web Jump” Description

The xSides product will contain a cartridge that will allow Internet searching, browsing, and display outside of the desktop screen. The user will be able to “jump” to areas of the Web without needing to be online, or a browser running. Full navigation function is available to the user in a “full view” mode, which presents the Web in the largest display window ever developed.



Sample images of the various looks and feels

Figure 12

3.2: xSides Web Jump: Operations

Two Modes of Use: The xSides "Web Jump" bar will operate in two modes: Search and Browse.

Search:

The bar will first launch in Search mode. This mode will present a side of the bar that holds an entry field for the user to type in search requests, or URL Jump requests. There will be two search functions: Search and Ask. The Search function will launch the default browser, initiate the search using the default search engine, and display the results. Ask will accept plain English language requests and post results.

The user can jump to any location by typing in a standard URL request with either "http:" or "www." at the beginning of the request. If the user presses the Enter key, the program will recognize this as either a search or jump request, and initiate the action.

A "Full View" button will change the look of the bar, so that full Web navigation functions are displayed.

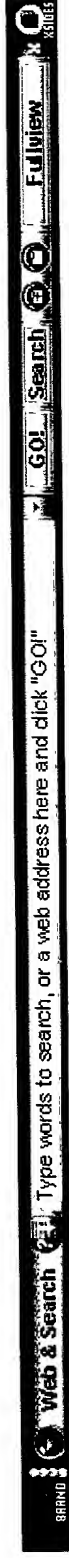


Figure 13

"Browse" with "Full View"

When the "Full View" button is pressed, the right-hand side of the xSides bar will display a similar row of browser function buttons. These will display navigation tools that lets the user move easily around the Web. These functions are further described in the "Appearance" section.

When the user types a request in the Input Field, the default browser will be launched in Full View, a pure Web display area with no navigation bars. The user will now navigate using the buttons in the xSides area. Clicking on the Full View button again allows the user to return to Search View.



Figure 14

Note: All Example xSides Layouts: Specific location of functional buttons may vary according to partner requirements

3.3: xSides Web Jump: Appearance & Action Requirements, Search Mode

Designed to provide quick jumps to the Internet, the user will be able to type in several types of requests and then jump out to the Net to have the action immediately taken. The first face of the Web Jump cartridge the User sees is "Search Mode". The following lists the functions accessed on the Web Jump cartridge.

- | | |
|-----------------------|--|
| 1. "Title" | This is your xSides browser. |
| 2. "Rotator" | This takes the user to the Search mode. |
| 3. "Input field" | This is where the user types in all searches, queries, and URLs. |
| 4. "Search" | This button performs a search on the information typed into the input field. |
| 5. "Go!" | This takes the user to the URL location typed into the input field. |
| 6. "Add to favorites" | User clicks this button to put the Web page currently in use into a Favorites or Bookmarks folder. |
| 7. "Open Favorites" | Brings up the folder with favorite (bookmarked) sites. |
| 8. "Full View" | This launches the browser in Full Screen Mode, and toggles Full View off as well. |

- | | | | | | | | |
|----|----|----|----|----|----|----|----|
| 1. | 2. | 3. | 4. | 5. | 6. | 7. | 8. |
| | | | | | | | |

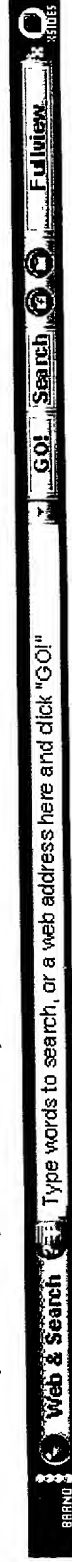
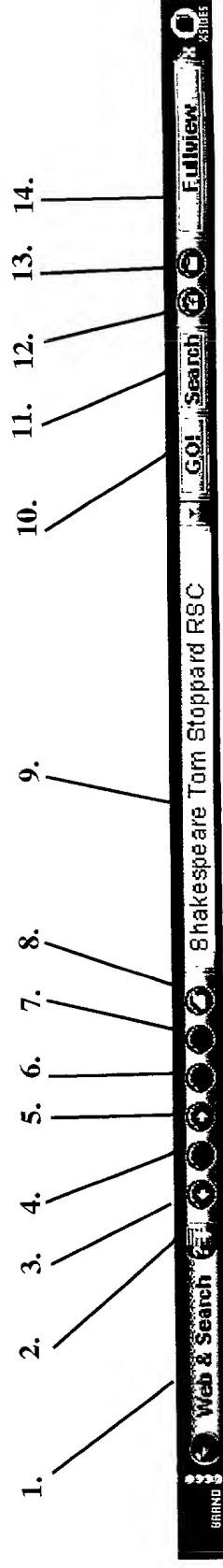


Figure 15

Note: Example xSides Layout: Specific location of functional buttons may vary according to partner requirements

3.4: xSides Web Jump: Appearance & Action Regs, Browser Mode

1. "Title"
This is your xSides browser.
2. "Rotator"
This takes the user to the Search mode.
3. "Back"
This button takes the user back to previous Web page
4. "Stop"
This button stops whatever Web access action is in progress.
5. "Forward"
This button takes the user forward to previously viewed, but "next" web page
6. "Reload"(Refresh)
This reloads, or refreshes, the currently viewed Web page.
7. "Home"
This button takes the user to their designated home page.
8. "Print"
This button provides browser print functionality.
9. "Input field"
This is where the user types in all searches, queries, and URLs.
10. "Search"
This button performs a search on the information typed into the input field.
11. "Go"
This takes the user to the URL location typed into the input field.
12. "Add to favorites"
User clicks this button to put the Web page currently in use into a Favorites or Bookmarks folder.
13. "Open Favorites"
Brings up the folder with favorite (bookmarked) sites.
14. "Full View"
This button changes the mode to Full View or Regular browser view.



3.5: The xSides Web Jump: Behavior Guidelines

Starting up:

xSides' Web Jump will work with the user's default browser application. The user will launch the xSides Web Jump while in one of three states:

1. Offline, and no browser running
2. Online, and no browser running
3. Online, with another browser running

The xSides browser will launch the user's default browser and will mirror its functionality consistent with the user's understanding and expectation. Therefore, Web Jump will tie into the information stored on the system registry files, or in the shared data files of the user's default browser.

Scenario #1: The user is working offline. With the xSides browser displayed on the bottom of the screen, the user types a question into the input field and hits Search. The logon sequence will begin and a Full View Web window will fill the screen. The user will then navigate by using the controls on the browser bar. (The user will actually be using his default browser functionality, even though its recognizable function bars will not be visible.)

Scenario #2 - The user is online and has no browser running. Same as above, except no logon sequence.

Scenario #3 - The user is online and has browser running. xSides browser will duplicate functions of the Web browser, including Favorites, Home, and other user oriented information. The user will be able to access the same information, in the same display format, by clicking on the same button in each area.

The xSides "Favorites" area will read files from the user's default browser. As new URL links are added to the user's default browser, they are also added to the xSides Favorites section, and visa versa. Clicking on the Favorites button in the xSides Browser area will bring up the same dialogue box as those in I.E. or Netscape.

A user's start page, the place he goes to first when surfing the Web, will be shared between the xSides browser and the user's default browser settings. xSides Browser will read the registry to deliver the same URL location.

When the mouse cursor rests over a button, a pop-up window appears. Pop-up Help may contain cascading menus, descriptive text, etc.

Handling Favorites

Handling "Home"

Pop-Up Help

[illegible]

Switching Between Full View and Various Sizes of the Regular Browser:

The user will be able to switch between viewing default browser in Full View mode or regular browser display. Both the Search bar and the Browsing bar will have a A Full View button displayed on the right of the Input Field. When the user clicks on this button, the screen will expand so the entire viewing area will be filled with the Web window, without any borders, toolbars, navigation buttons, etc. The user clicks the button to toggle between this Full and regular Web views.

- 1. xSides Search mode with Standard browser functionality.** With Search mode the user's default browser will function normally. Users will now simply have the choice of having a Full View mode, in addition to the standard maximize and minimize modes within Windows. When the user is in the Search bar and clicks the Full View button, xSides will rotate to the Full View browsing controls where the user can navigate the web with an expanded Browser viewing area.



- 2. xSides Full View mode with Expanded Browser viewing area.** When the user wants to return to their standard browser viewing mode they click on the Full View button again and it will rotate the control bar back to the search bar and display their default browser.

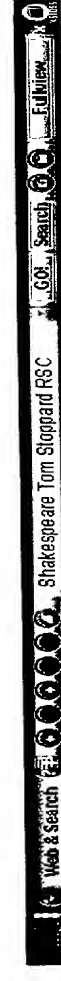
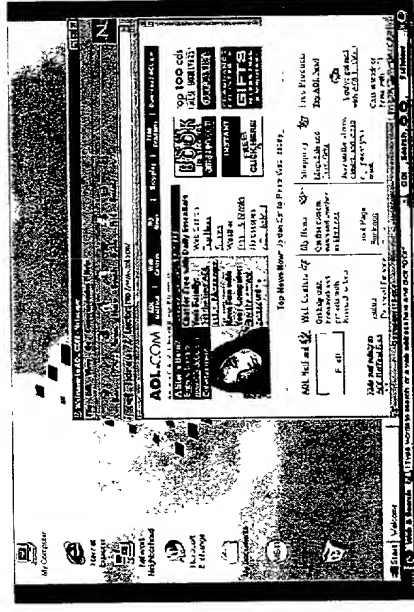
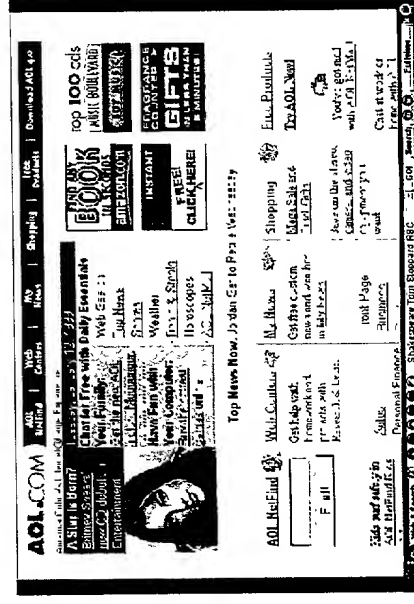


Figure 16

Draft -Confidential The Pixel Company Copyright 1999 All rights reserved Patent pending
No portion of this document may be reproduced without permission



(Standard browser state)



(Full View browser state)

Appendix C

PIM side for the xSides bar

Rev. 10

(Personal Information Manager)

Product Specification

December 7, 1999

Project Manager – Bronwen Matthews
The Pixel Company

1 Functional Specification

1.1 Overview

This project is designed to convert the calendar and other appropriate features/tools from the MySpace OnlineApartment; plus new tools, into an extended (portal) side for the xSides bar. These features/tools fall under the category of Personal Information Management. The title of the actual side is still up in the air at this point.

1.2 Implementation Schedule

Phase 1 – done/changed

PIM bar only (no portal) –

Phase 2 - done

Phase 2 provides for a PIM bar with Calendar, Contacts and To Do list only utilizing the portal area, (ie. Partial functionality for the portal area).

Phase 3

PIM bar with full functionality plus,

Make the following revisions to the current calendar:

- Add wedding and anniversary icons to the calendar.
- Add ability to schedule repeating events on specific dates, for ex., the 1st of every month and one day per year, on a specific date (birthdays and anniversaries, etc.) Events like birthdays and anniversaries could just auto-repeat. (add to pull down menu?)
- Change events that recur every other week to start on the first week they are scheduled.
- Enable non-time-specific events to appear in the cream box, maybe with a different colored background. (part of a later phase)
- Make Help window resizable.

Phase 4, ?

Expand the Daily News button into an individual face for the PIM. The user would roll to this other face. (not currently outlined in this spec., not currently decided upon.) Replace Daily News button with: Expense Report button or Web-based Word Processor button.

Phase 5, ?

Make the calendar able to used offline (without an Internet connection).

1.3 End User Requirements

1.4 Hardware

1.4.1.1 Client Side

- Internet access and provider
- Min Processor: 486DX/66 PC (Pentium® or higher recommended)
- 3MB of free hard drive space
- 20MB of RAM; (24MB of RAM recommended for Win95)
- 24 MB of RAM (32 MB recommended) for Win98
- Monitor: VGA (with standard resolutions)
- Mouse and Keyboard support.
- Recommended for single-monitor systems
- Direct X Support (installer will install if needed)
- Modem

1.4.1.2 Server Side

- WWW server
- SQL database server

1.4.2 Software

1.4.2.1 Server Side

- Database(s) to support the calendar, Distributor and Instant Alert.
- SQL 6.5 or 7.0
- IIS 4.0+
- User Registration

1.4.2.2 Client Side

- Windows 95 or 98 or Windows NT Operating System
- Install detects whether client system needs to have additional support software installed.

1.5 End User Features in general:

The user will download a single xSides side that is the PIM or will download a prepack that includes the PIM. By enabling users to store and access personal information, it is hoped that the PIM side will increase stickiness for the xSides bar.

Users will be able to access their own personal address book through this side as well as maintain their own personal calendar. With the use of Instant Alert, they can, if they wish, be notified in advance of events listed on their personal calendar either through email or a desktop pop-up instant alert window. They will also be able to access a calculator; a portal for writing and finding memos – notes the user generates about any topic; links to headlines from our Daily News, the magazine rack from MySpace Online; and an Export/Import feature, which will allow for the exchange of information as it relates to the Address book, calendar, To-Do list, and Memo features.

1.6 Deliverables

1.6.1 Client Side

1.6.2 Server Side

1.6.3 Installer

2 UI Analysis

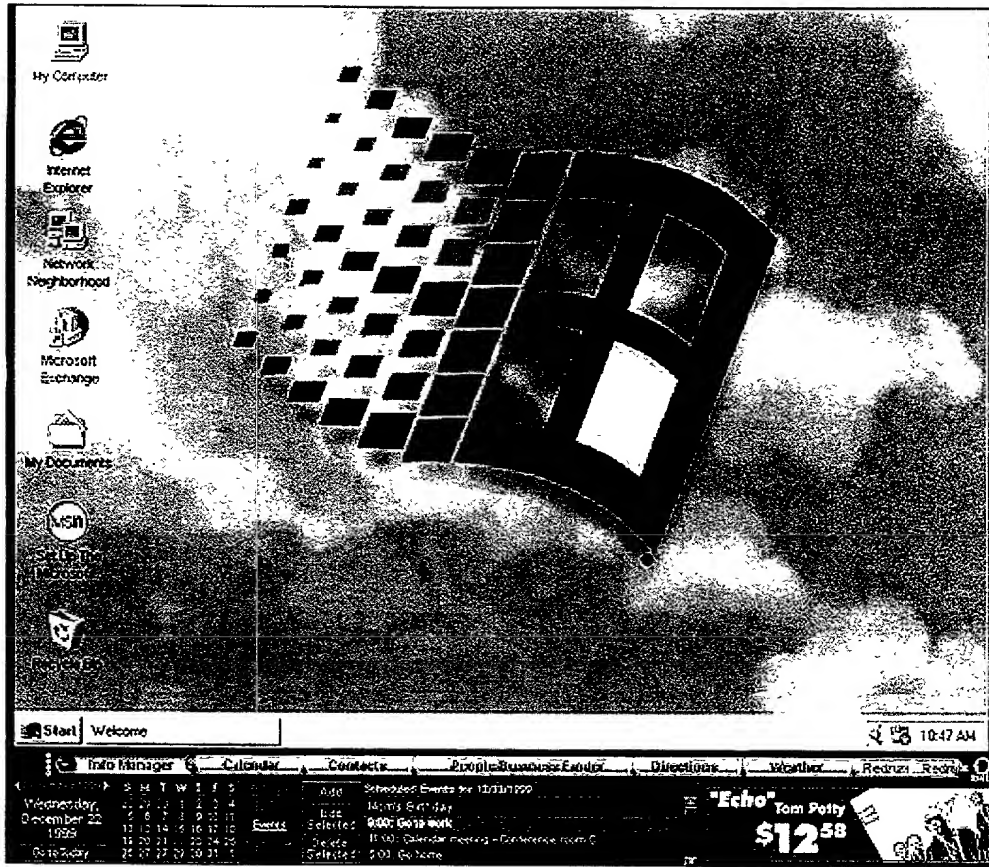
2.1 PIM side for xSides Bar: (from left to right)

2.1.1 PIM/Info Manager side buttons:

The category title area on the left-hand side of the side will contain the title: “PIM” or “Info Manager” or another name to be specified. The Info Manager side includes the following buttons from left to right: Calendar, Contacts, Calculator, Memo, Daily News, Newsstand, and Import/Export. If a user clicks on a button, it causes the associated portal to appear in the portal area beneath the bar.

2.1.2 Calendar button:

The first button to the right of the category title area will be labeled “Calendar”. When the user clicks on the Calendar button the following information will appear from left to right, in the extended portal area below the bar:



Calendar Portal showing the Scheduled Events list.

2.1.2.1 Current Date.

The current date is displayed on the left side of the Calendar portal area.

2.1.2.2 Go To Today:

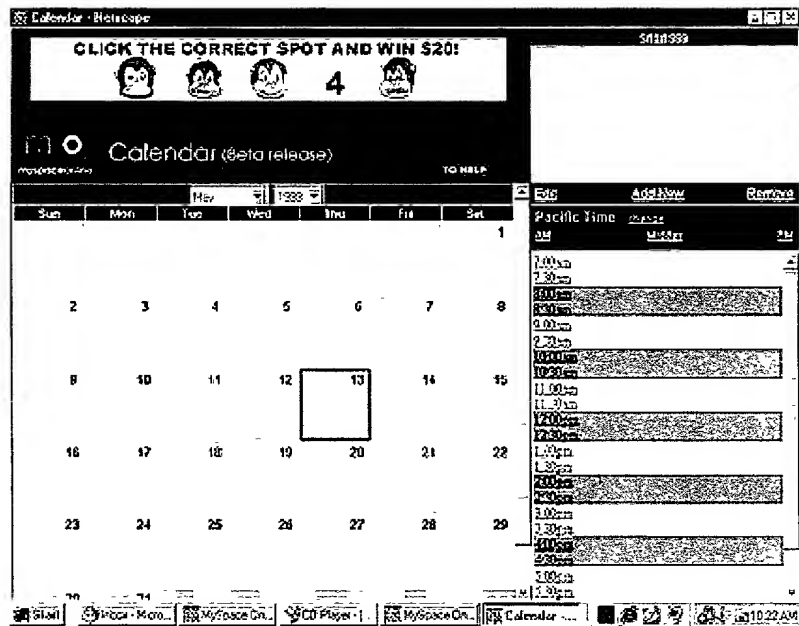
Clicking on the Go To Today button on the Calendar portal causes the current day's date to highlight on the monthly calendar and presents the current days scheduled events in the portal area to the right.

2.1.2.3 Monthly calendar:

A small current monthly calendar, with the current date highlighted appears to the right of the current date.

2.1.2.4 Full-Size Calendar:

Clicking on the Full-Size link causes the existing calendar (from MO), to launch in a browser window above the bar for a more extensive view of events.



Full Size (existing) Calendar.

2.1.2.5 Events:

Clicking on the Events link causes the Scheduled Events list to appear to the right, in the portal area. This list presents the information from the hourly section (blue and white section), of the users' full view calendar (shown above). This section is scrollable and presents controls to ADD, EDIT SELECTED and DELETE SELECTED. If the user clicks on ADD, they will be presented with the ADD EVENT/APPOINTMENT page of the existing calendar. The user selects an item to edit or delete, by highlighting it. Clicking on DELETE SELECTED will automatically delete the record of this event. Clicking on EDIT SELECTED will pop up the EDIT EVENT/APPOINTMENT page of the existing calendar, showing the current information for that particular event. The user makes any necessary changes and resubmits the event by clicking on ADD THIS EVENT NOW.

To DELETE an item on the To Do list, the user will first need to select it by highlighting it. Clicking on EDIT SELECTED or ADD presents the EDIT or ADD TO, TO-DO LIST pop up dialogue box. The user adds information or makes changes and clicks on SUBMIT.

TO-DO LIST

Task:

Priority:

Breakdown:

1:

2:

3:

4:

5:

6:

7:

8:

9:

10:

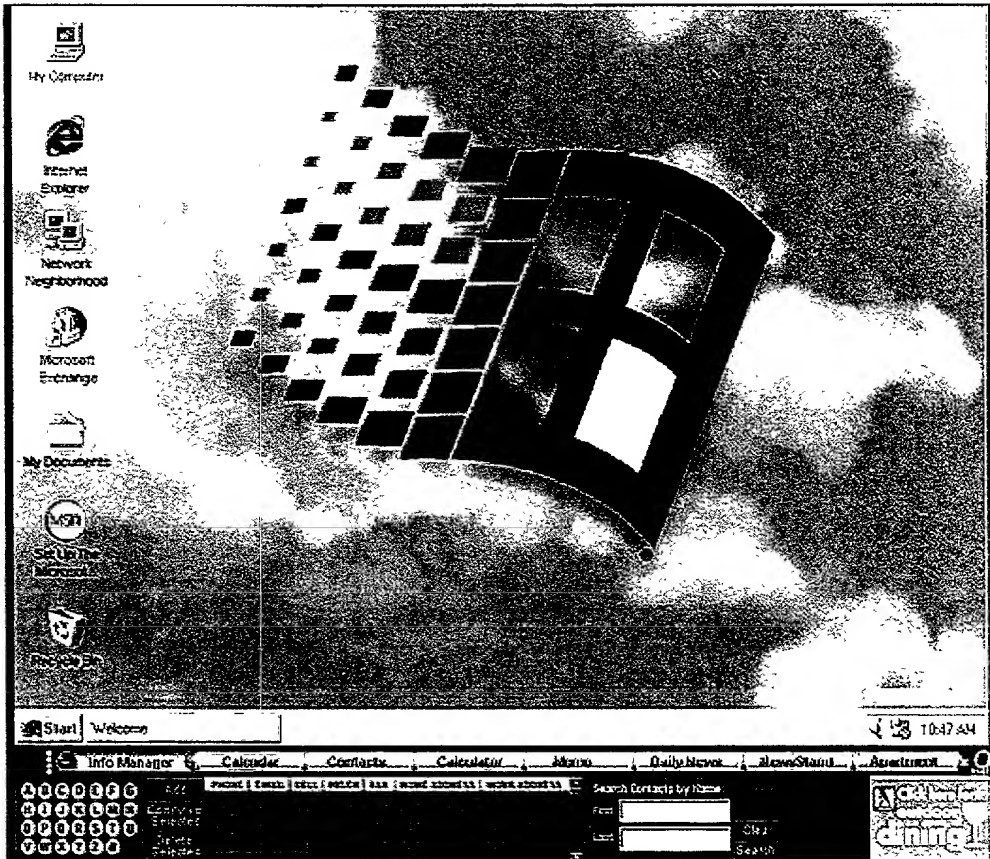
Add/Edit To-Do List interface

2.1.2.7 Small Ad Space:

To the right of the To-Do list is a variable-sized banner ad space. This banner ad space is present for all the portals for the PIM, but may vary in size due to the size of the content windows. Size of the presented ad is dependent on the resolution of the end users screen.

2.1.3 Contacts Button:

The next button on the bar to the right of the Calendar button is the Contacts button. When the user clicks on this button it will launch their address book in the portal area. They will see the following features, from left to right, in the portal area below the bar:



Contacts Portal

2.1.3.1 An alphabetical keypad for selecting listings:

The user clicks on a letter or the # sign to see listings for people whose last name begins with the selected letter or a number. These names are displayed in the scrollable box to the right of the alphabetical listing.

2.1.3.2 ADD, EDIT SELECTED and DELETE SELECTED Buttons:

To add a name, the user clicks ADD. This presents the current calendar's ADD/CHANGE form. They then fill in the information and click on SUBMIT ENTRY CHANGES. To EDIT an entry, the user selects a listing in the box to the right and then clicks on EDIT SELECTED. This brings up the Add/Change form for that listing. To DELETE a listing, the user selects the listing and then clicks on DELETE SELECTED.

[illegible]

2.1.3.3 Search by Name

The next window presents the Search by Name feature. This is part of the functionality of the existing calendar. The user types in a first or last name and clicks on search to search their directory of contacts. The results appear in the scrollable listing area to the left.

2.1.3.4 *Contacts Help*

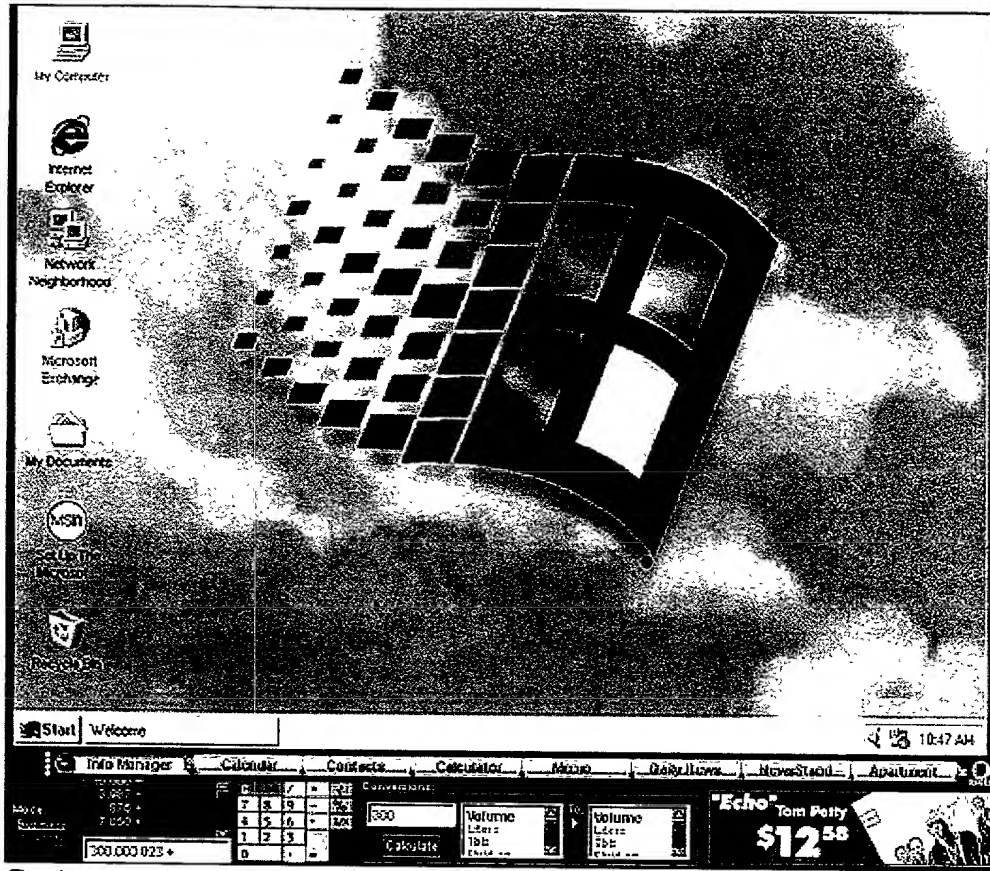
Links to the Help for the Address book. (***This will need to be modified for this version of Address Book/Calendar.**)

2.1.3.5 Clear

Clicking on this button clears the current search.

2.1.4 Calculator Button:

The next button to the right of Contacts is the Calculator button. There are two display modes for the calculator: Business and Scientific. The user may toggle between the two modes, using the two links on the left side of the portal.



Business calculator mode.

2.1.4.1 **Business calculator mode**

The Business calculator mode lists the entered numbers in a column in the display area on the left side of the screen and keeps a simple running tally, shown in red on the display area.

2.1.4.2 **Onscreen Keypad**

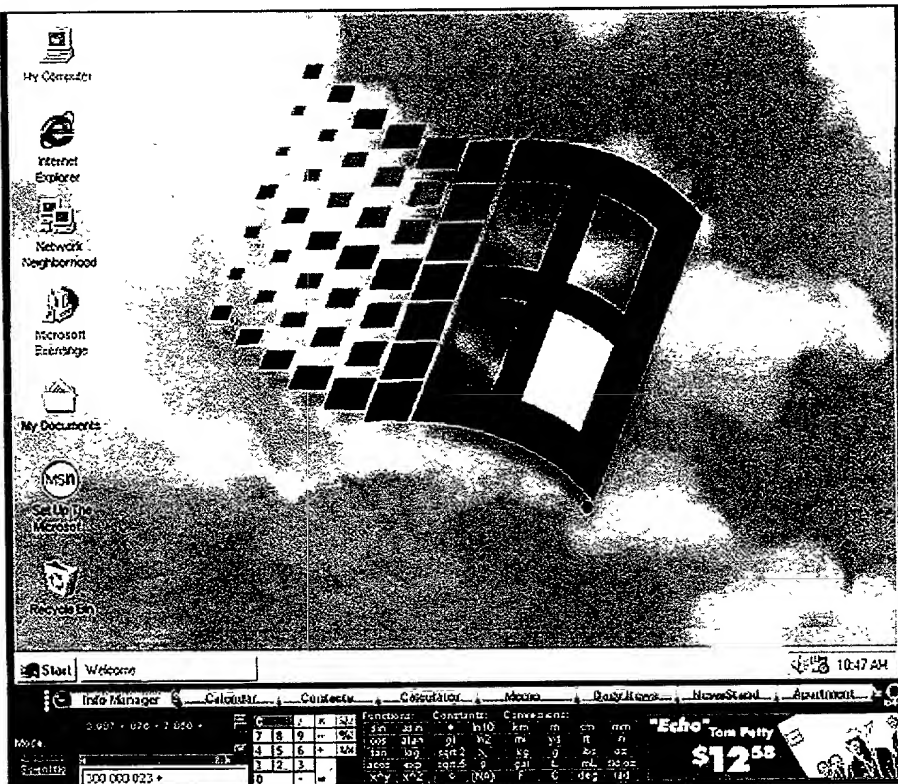
To the right of the display area is an onscreen representation of a traditional keypad. The user can enter numbers and functions here by clicking on buttons on the onscreen keypad or by using their actual keyboard.

2.1.4.3 **Conversion section**

The user may enter a number in the window in the conversion area and then select the unit of measurement that they are converting from, and the unit of measurement that they are converting to. They then click on the Calculate button and the answer will appear in the white window in the conversion area.

2.1.4.4 Scientific calculator mode

The Scientific calculator mode displays actual calculations in a scrollable list. Multiple functions can be shown in the display area, and be revisited by scrolling through the display area.



2.1.4.5 Onscreen Keypad:

To the right of the display is an onscreen representation of a traditional keypad. The user can enter numbers and functions here by clicking on a number or function on the onscreen keypad or by using their actual keyboard.

2.1.4.6 Additional Functions, Constants and Conversions:

The next section to the right contains the most used Functions, Constants and Conversions for scientific calculations. The user selects a number in the white sum display area on the left and performs a function on it by selecting from Functions, Constants or Conversions.

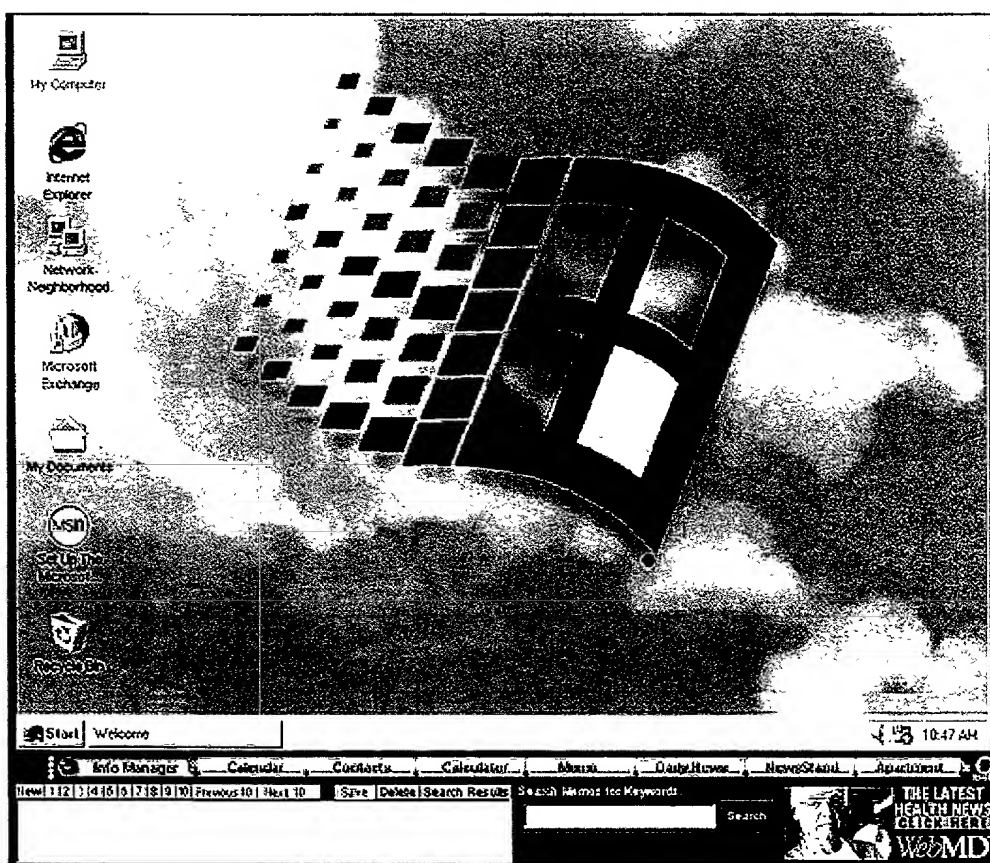
2.1.4.6.1 **Functions** work the same way that they do for the Business Calculator.

2.1.4.6.2 **Constants** can be applied to calculations or can be selected and presented by themselves. Clicking on a constant alone, will present the constant in the white display area.

Parameter	Value	Unit
α	0.001	
β	0.001	
γ	0.001	
δ	0.001	
ϵ	0.001	
ζ	0.001	
η	0.001	
θ	0.001	
ι	0.001	
κ	0.001	
λ	0.001	
μ	0.001	
ν	0.001	
ξ	0.001	
\omicron	0.001	
π	0.001	
ρ	0.001	
σ	0.001	
τ	0.001	
υ	0.001	
ϕ	0.001	
χ	0.001	
ψ	0.001	
ω	0.001	
Ω	0.001	
Θ	0.001	
Υ	0.001	
Φ	0.001	
Ψ	0.001	
Ξ	0.001	
\Omicron	0.001	
Π	0.001	
\Rho	0.001	
Σ	0.001	
Υ	0.001	
Φ	0.001	
Ψ	0.001	
Ξ	0.001	
\Omicron	0.001	
Π	0.001	
\Rho	0.001	
Σ	0.001	
Υ	0.001	
Φ	0.001	
Ψ	0.001	
Ξ	0.001	
\Omicron	0.001	
Π	0.001	
\Rho	0.001	
Σ	0.001	
Υ	0.001	
Φ	0.001	
Ψ	0.001	
Ξ	0.001	
\Omicron	0.001	
Π	0.001	
\Rho	0.001	
Σ	0.001	
Υ	0.001	
Φ	0.001	
Ψ	0.001	
Ξ	0.001	
\Omicron	0.001	
Π	0.001	
\Rho	0.001	
Σ	0.001	
Υ	0.001	
Φ	0.001	
Ψ	0.001	
Ξ	0.001	
\Omicron	0.001	
Π	0.001	
\Rho	0.001	
Σ	0.001	
Υ	0.001	
Φ	0.001	
Ψ	0.001	
Ξ	0.001	
\Omicron	0.001	
Π	0.001	
\Rho	0.001	
Σ	0.001	
Υ	0.001	
Φ	0.001	
Ψ	0.001	
Ξ	0.001	
\Omicron	0.001	
Π	0.001	
\Rho	0.001	
Σ	0.001	
Υ	0.001	
Φ	0.001	
Ψ	0.001	
Ξ	0.001	
\Omicron	0.001	
Π	0.001	
\Rho	0.001	
Σ	0.001	
Υ	0.001	
Φ	0.001	
Ψ	0.001	
Ξ	0.001	
\Omicron	0.001	
Π	0.001	
\Rho	0.001	
Σ	0.001	
Υ	0.001	
Φ	0.001	
Ψ	0.001	
Ξ	0.001	
\Omicron	0.001	
Π	0.001	
\Rho	0.001	
Σ	0.001	
Υ	0.001	
Φ	0.001	
Ψ	0.001	
Ξ	0.001	

2.1.5 Memo:

The fourth button on the PIM bar launches the Memo feature in the portal area. This is a tool the user can use to write short notes about any topic. It could be a list of gifts to buy, a book they heard about, design ideas, a phone number they need to jot down, etc. The Memo portal from left to right includes the following:



Memo portal

2.1.5.1 Memo Input/Retrieval area:

Each memo has a number (like a page number) associated with it, for a total of 50. The numbers appear across the top of the input area beginning with New. The user can select a number from the list across the top of the Input/Retrieval area to see a specific memo, by choosing from the numbers that appear or by clicking on Previous 10 or Next 10.

When the user first goes to the memo portal, a new (blank) memo appears.

The white area is the space where the user enters their memo content.

2.1.5.2 Save/Delete:

After a user types a memo in the Input area, they must click on Save to save the memo.

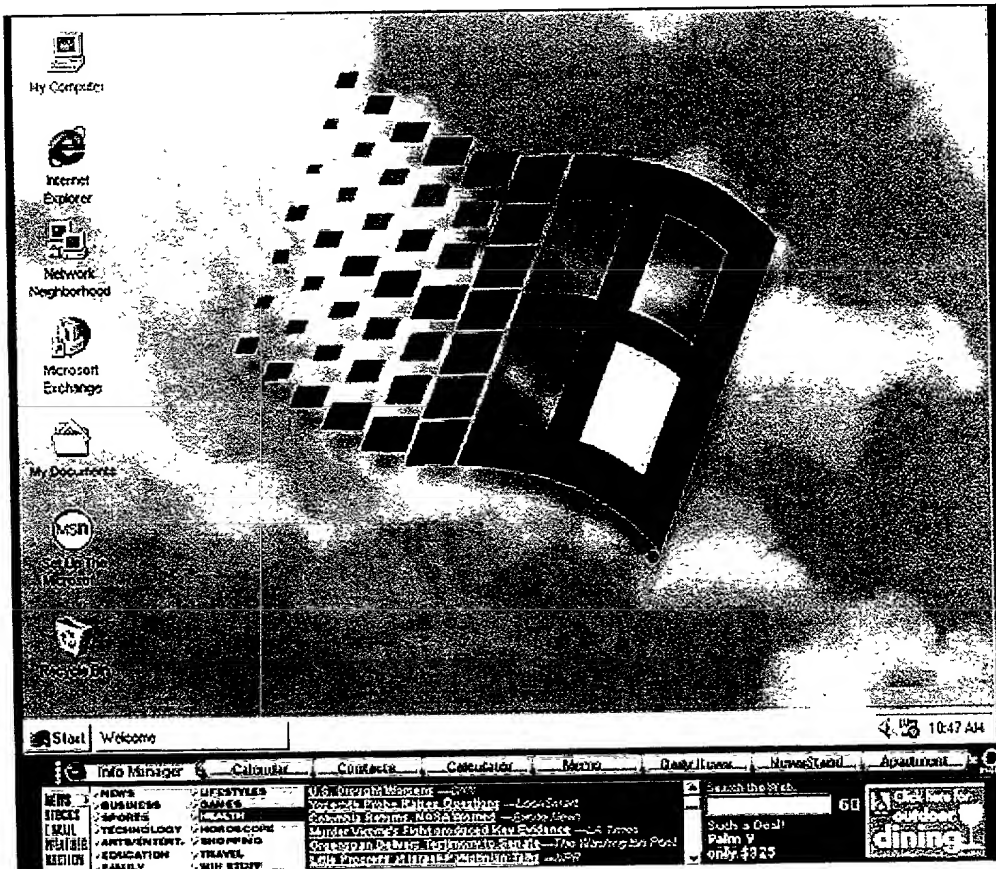
Users can Delete a memo by selecting a specific memo and clicking on the Delete button.

2.1.5.3 Search Area:

A user can also retrieve a memo using the Search area. They simply type a word they remember using in the memo, into the Search box and click on the Search button. The result of the Search appears in the white area. They use the Search Results button to page through various pages of results.

2.1.6 Daily News:

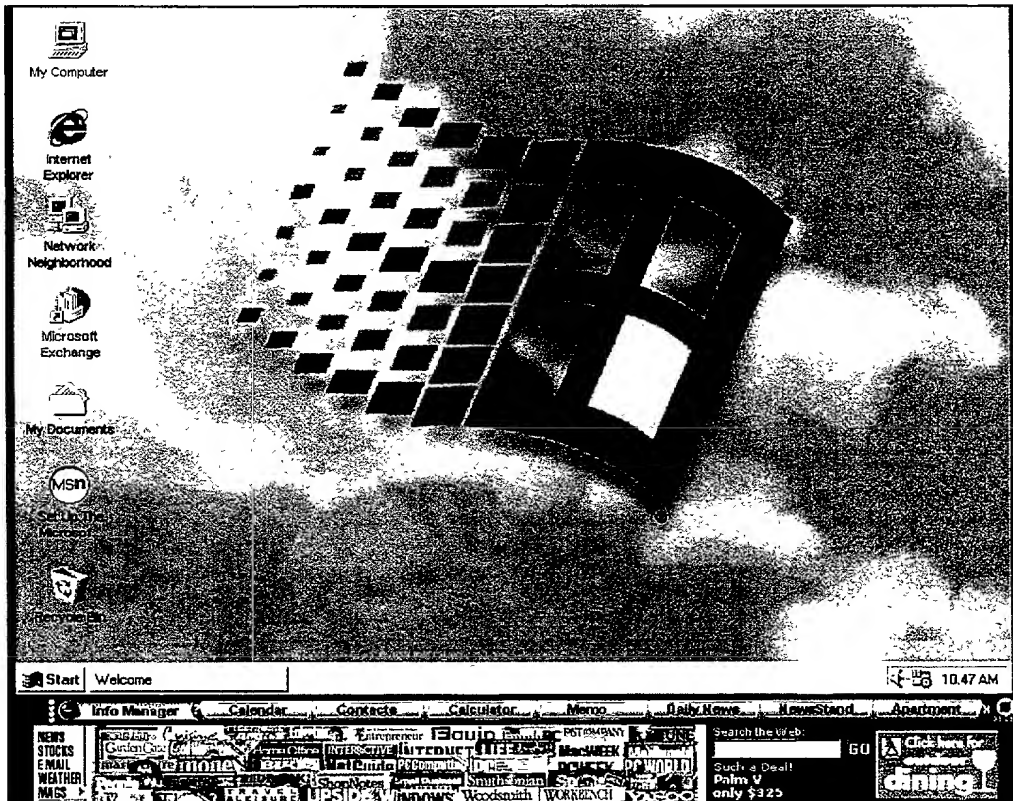
The fifth button on the bar is Daily News. Clicking on this button presents the headlines from the current MySpace Online Daily News to appear in the portal area. We already have this functionality working for the generic beta so all that may be required here is a slight design change to make this feature appear consistent with the rest of the PIM portals. The Daily News portal will contain a menu listing each of the major sections of the Daily, such as News, Sports, Weather, etc. (Please disregard the beige menu on the far left. This will not be included.)



Daily News Portal

2.1.7 NewsStand:

The sixth button on the PIM side is the NewsStand button. Clicking on this button presents a graphic of the current MySpace Online newsstand (also called magazine rack) in the portal area. Each magazine links directly to that publication's online ezine or publication.



Newsstand portal

2.1.8 Import/Export:

The last button on the bar, on the far right is Import/Export (not Apartment as shown in most of the bitmaps). Clicking on the Import/Export button on the bar will launch an Import/Export interface in the portal area. The Import/Export feature lets you enter (Import) scheduling, address and memo information that the user may already have in another program into the xSides PIM, without re-entering information one entry at a time. It also lets you take (Export) the same type of information out of the PIM as a text file that you can save on your computer, print out, or enter into another device or program. The user will choose which action they would like: import or export and will select which PIM item they wish to Export or Import and which file formats they would like to use. Initially at least, this feature will import or export tab delimited, or comma

delimited text files and xml file formats only. The user would have the option to Import/Export information for/from the calendar, contacts, memo, and to-do list.

2.1.8.1 Functionality:

Importing:

A user's file is imported to our server, where it is parsed and put into the PIM database. We will use an ISAPI dll to grab the file off of the client machine and upload it using the upload code that has been written for the Communications layer for xSides/AllSides. Since there is not a one to one field correspondence from the PIM to other applications that may be used for importing/exporting, for example, Outlook, the user may have to manipulate the data to make it appear in the correct text format

- 2.1.8.1.1 To Import: Importing into the PIM takes two steps: the user must export information from their original program or device and then import it into the PIM.
- 2.1.8.1.2 Preparing a file to be imported into the PIM: To prepare a file to be imported into the PIM, the user must export it from their original program or device and save it as a text file, using the file extension .txt. As part of exporting, the user will need to select the characters to be used in between fields: comma, semicolon or tab.
- 2.1.8.1.3 The user may have the option to export field names on the first line. This is entirely optional but will make it easier to identify fields when importing. If they chose this option during Export, they must put a check next to "Does this include titles?" on the Import portal on the PIM.
- 2.1.8.1.4 Importing into the PIM: In the "Import" box, under "File Name:" on the Import (lefthand) side of the portal, the user enters the name of the file they would like to import into the Calendar, Address Book, Memo, or To-Do List. If they don't know the file name, they can click the "Browse" button to search for it.
- 2.1.8.1.5 Under "File Destination:" they indicate where they want the information imported to by selecting the button next to the destination.
- 2.1.8.1.6 Under "File Type:" they indicate the type of the file by selecting the button next to the method used in their file to separate the fields of information. They can open their file in Notepad and see which punctuation is used (or if the file is in XML) to separate the fields.
- 2.1.8.1.7 Click "Next." They will see their information displayed in separate fields. Use the drop-down boxes above the fields to choose the appropriate label for each field; e.g., if the information contained in the first field is start dates of events, select "Start Date" from the drop-down menu. Choose the correct label for each field of information or choose the Ignore label.

- 2.1.8.1.8 Click "Show Me" to see the results of labeling in the box below. Check the information to make sure all the fields have the appropriate label. If not, go ahead and make additional changes to the information in the drop-down boxes. Click "Cancel" to cancel you import. Click "Submit" if all information and labels look correct and your data will be imported.
- 2.1.8.1.9 To Export: In the "Export" box, under "File Type:" select the type of file you'd like to export. Check the "Include Titles?" box if you'd like to export the titles of the information fields in your file.
- 2.1.8.1.10 Under "Export Type:" indicate the type of punctuation to use to separate the fields of information (or choose XML) by selecting the appropriate button. To import this data to another scheduling system, choose the same punctuation (or XML) that that program requires.
- 2.1.8.1.11 Click "Go" and a dialog will pop up asking where on your computer you would like to export (save) the file to. Choose a location and the file will be saved in that location.

2.2 Future Possible Features:

If the Daily News is spun off as its own individual face of the PIM bar, this button could be replaced with an Expense Report Button or a Net-based Word Processor button.

2.2.1 Expense Report/Checkbook Register:

The PIM could include an Expense Report button. Clicking on this button would cause a simple form that we create, to pop up either in the portal area, if there's room, or full screen, above the bar. Our form would export to the leading templates for expense reports such as Excel. We could try to work with Quicken and MS Money as well.

2.2.2 Print Feature:

We would also like to include the ability to print any type of PIM information. We may call for a Print button on the bar, which would function the same way as Import/Export in that clicking on this button would launch an interface that would allow the user to choose what they want to print. Or there may be a print button added to all of the relevant portal/pages.

2.2.3 Email:

We could add an Email button to the PIM bar. When the user selected Email, we could present the users' pop 3 email inbox or Exchange inbox. If the user didn't have email, this button would be dead (not ideal). When the user initially clicked on this button, a small interface would pop up asking them to identify their existing email service.

3 Technical Analysis

3.1 Class Analysis of Features/Requirements

3.1.1 Client Side

3.1.1.1 Processes

CALENDAR: While all of the information storage and retrieval are done on the server side of the application, all presentation and formatting of the data is handled on the client side. This method reduces the traffic across the network, eliminating net traffic slow downs. Entire blocks of data are retrieved at once, allowing the client-side engine to deliver a variety of pages from a single download. The full-sized calendar and the portal calendar are actually two separate applications that draw from and write to the same server data source.

All of the GUI presentation is handled client-side, including the calendar-view itself. When the users enters the calendar, it retrieves the current local date and formats itself correctly. This engine is set up to handle dates until the year 2010, but can easily be extended as needed.

TO-DO LIST: Although this appears on the same side as the calendar, it is actually treated as a separate application. The To-do List uses a similar server-client set up that calendar uses, but because the To-do List has a more limited storage capacity, a users entire data set is retrieved at once. This allows the application to limit the net traffic, connecting to the server only when changes to the stored data are requested.

ADDRESS BOOK/CONTACTS: Once again, this application uses a similar breakdown of server to client workload. The client handles all the data formatting and presentation, while the server only handles data storage and retrieval. All the data is retrieved in blocks according to the user request. For example, if a user clicks on the letter A, all entries that match the query are return and shown to the user. A search feature in this application can request modified data sets based on names or letter combination in the record set.

CALCULATOR: The client-side engine handles all processes for this application. The engine parses the user request based on what buttons are presses, then presents the resulting data to the user through simple html modification. Specialized calculations are handled by individual functions, tailored to match needed data input and returned.

MEMO: Similar to the To-do List in its data structure and server-client workload breakdown. The Memo application does require the specialized formatting that the To-do List does, so much of the client-side presentation is WYSWIG, displaying the data exactly as it gets stored in the database.

NEWSSTAND: This is an html page, modified to fit the needs of the portal and PIM.

DAILY NEWS: Current headlines from leading news sources are presented, on a daily basis, across 14 categories, including Top News, Entertainment, Technology, Business, Sports, and Lifestyles. Clicking on the headline text will make the report or feature available for viewing.

3.1.1.2 Diagrams

3.1.2 Server Side

3.1.2.1 Processes

EXPORT: After the user clicks on the export button on the PIM and chooses one of our apps.(calendar/addressbook/memo/ToDo list) to export, a text file will be available to them to save on their machine. To do this, an asp page(s) needs to have access to all 4 tables though stored procedures. The .asp page then writes the database data for that user to the client browser. The header to this html page should be set as a text file so that the user will get a dialog asking where to save the page.

USER DEFINED VARIABLES: In all cases the user will be passing the userID (GUID), a number indicating which application from which to export, and a formatting code indicating how to display the data.

The GUID needs to be first converted to base64 and there is an existing VBScript function to do this. The formatting will be used to indicate whether the data is to be displayed with spaces or commas between fields or in xml.

For the Calendar, the user needs to pass a beginning and end date indicating the period of events they want returned. For the Addressbook a alphabetical character is passed or a not alphabetical Character indicating they want all their entries.

CODE DESIGN: This could be implemented in 2 .asp pages. The backend page contains the functions that call the stored procedures. Also it needs to contain the GUID conversion code that converts the GUID into a base64 for number. This is stored as a base64 number to save on space in the database. The main page would use 'if' statements to call the unique code appropriate for each application including writing out the data to the page. The logic for the main page is to get all the variables, convert the GUID, go to the appropriate section based on the application number. Then in that section call the function that runs a stored procedure. Each stored procedure will return a recordset. Then Loop through the recordset and display the data based on the format code using the 'response.write' method (See Below) for XML. E.G. if the comma code is passed in then display commas between fields. All display data should include the titles. Titles will be displayed first and this can be done in the same stored procedure

The format for displaying the return data will be different for each application. The Calendar will display user defined data, i.e. start date, end date, type of event, description, location. and notes. The address book will display alphabetically the first, mid, last names, street, city, zip code, work phone, home phone, fax, cellphone, and email. The ToDo list will display the item and then its subitems if any possibly indented to indicate these are subitems and then the priority number. The memo will display a sequential number and then the memo data. These can be displayed

Chronologically because they are in the database with a timestamp and the stored procedure for the memo table can return the recordset as 'ORDER BY timestamp ASC'.

The returned page should set the header to 'Text'. This will allow a dialog box to show that will Permit the user to save the page wherever they wish on their drives.

XTML OUTPUT: As one of the output formats, it will look like this for the calendar:

<CALENDAR>

<TITLE EventType = Event Type StartDate=Start Date EndDate = End Date Description = Description, Location = Location Notes = Notes> </TITLE>

```

<EVENT EventType = Type StartDate=StartDate EndDate = EndDate Description =
Description. Location = Location Notes = Notes></EVENT>
</CALENDAR>
<ADDBOOK>
<TITLE Last=Last Name Mid= Initial FIRST = First Name CITY=city STREET = Street
STATE=State ZIP=Zip BIRTH = Birthday EMAIL = Email PHONE=Home WORK= Work
Phone CELL= Cell Phone PAGE=Page FAX = Fax COMPANY= Company
COStreet=Company Street SUITE=Suite COCITY= Company City COSTATE= Company
State COZIP = Company Zip></TITLE>
<ADDRESS Last=Last Mid= Mid FIRST = First CITY=city STREET = Street STATE=State
ZIP=Zip BIRTH = birthday EMAIL = Email PHONE=phone WORK= workphone CELL=
cellphone PAGE=page FAX = fax COMPANY= company COStreet=costreet SUITE=suite
COCITY= Cocity COSTATE= Costate COZIP = Cozip>
</ADDRESS>
</ADDBOOK>
<TODO>
<TITLE DESC=Task PRIOR= Priority></TITLE>
<TASK DESC= eventtitle PROIR= Prior></TASK>
<SUBTITLE DESC=SubTask SUBPRIOR= priority><SUBTITLE>
<SUBTASK DESC= eventtitle PROIR= Prior></SUBTASK>
</TODO>
<MEMO>
<MEMOTITLE TITLE=Title></MEMOTITLE>
<MEMOTITLE=Data></MEMOTITLE>

```

Comma Delimited format.
For the Calendar

PIM CALENDAR

Event Type, Event Description, Event Start Date, Event, End Date, Event Location, Event Notes
Meeting,Weekly update, 10/1/1999 8:00:00am,10/1/1999 10:00:00am,atWork, Bring handout

PIM ADDRESS BOOK

Last Name,Initial,First Name, Street,City,State,Zip,Birthday,Home Phone,Work Phone,
CellPhone,Fax,Company,Suite,Company Street,City, State, Zip

Pleat,,W, Geoffrey, Seattle,Wa,98125, 5/10/1957, 2065551212,2063361625,2065551212,,xsides 821
Second Ave.,Suite 1600,Seattle, Wa, 98102

PIM TODO

Task, Priority
Mow grass, 5

PIM MEMO

Memo Data
Call IRS at 8005551212

For Space Delimited leave the commas out.

4 Marketing

4.1 Product Distribution:

4.1.1 AllSides Download Site:

The PIM side will be featured on the AllSides download site, both on the home page and on an individual description page on the AllSides site. Users will be warned that they must have xSides 2.2 with merge capability in order to merge the PIM into their xSides bar.

4.1.2 AllSides Download Interface as part of xSides version 2.2

The PIM side will be listed as one of the available sides on the AllSides Download interface, which is part of the Quick Start Help/preferences pop up. (See AllSides spec.) User can choose to download and merge the PIM using this simple interface.

4.1.3 Product Promotion:

The PIM will also be promoted in our press releases and may also be promoted in our tradeout ads.

CONFIDENTIAL

[illegible]

This is a quick description of xSides Conversion and Merge - drawn from the development spec for xSides:

1.1 End User Features

Version 2.2 of xSides should accomplish and provide the following:

(Please Note: the old data structure is contained in an ini, the new data structure is contained in a "Markup file" - inis will not go forward in xSides after conversion).

1 Conversion of Old Data Structures (MySpace 2.1) to New Data Structures (xSides 2.2+)

Conversion of old data structures will happen with a separate executable that resides in the same installed directory as the xSides executable, the new executable is called **converter.exe**.

Conversion process:

- a) Each and every time the xSides executable (pixels.exe) is launched it will check to see if there are any ini or mark-up files of an older version or format that need to be converted to the newest data format.
- b) If ini or mark-up files exist that need to be converted pixels.exe will call converter.exe to make the old data structures match the new data structures and write it into the new data file: **pixels.dat**

2 Merging data

To provide the ability to have dynamic xSides content the ability to merge new cartridges and portals into xSides will be added.

Merge process:

- a) After the xSides executable has performed any conversion of data files that may need to be done, it performs a further check to see if there are any newly downloaded pixels.dat files to be merged with the current pixels.dat. (Pixels.dat files contain data structures for xSides bar and portal areas.)
- b) Pixels.exe will merge the previously existing xSides data and the new xSides cartridge(s) data into a single data tree. And write the newly merged markup file (pixels.dat) in the appropriate install directory.
- c) Merging will happen at the Platinum level of xSides, such that if duplicate data at this level, Category Name ("Where" in the ini/markup), is present in merging bars, the newest data will replace previous data. Any unique data will be merged as a new Platinum category into the existing category list.

In All of these scenarios the following would be true:

- Merging will happen at the Platinum level.

Example:

If xSides Distributor A has an Entertainment Category, and xSides Distributor B has an Entertainment category, when the user already has distribution A installed then downloads distribution B, distribution B will replace distributor A's Entertainment category.

- The most recently downloaded distribution of the xSides bar will take precedence in the merging (replacing duplicate sides with most recently downloaded) - except if it is not the most recent version of the application.
- When Cartridges are the unique at the Platinum level between distributions - unique cartridges will be retained.
- Cartridges will always maintain the original associated distributor's logo.
- Deletion: will be allowed When the Marketplace version is available and installed by the user. Deletion will happen by User choice at the Side level after the application has been installed and they have connected to Marketplace through the xSides interface in Help/Pref/Map/Marketplace area. The user will be presented with a list of currently installed bars along side the other bars the user can install, This dialog also provides that the user can select from the list of currently installed bars and select to remove them.
- There are no limits to the number of cartridges the user may install to their xSides bar. In the case of xSides SDK versions - there should be a statement that the could be a performance hit if they have more than # installed.

- [illegible]

xSides

xSides' breakthrough technology puts more usable space on your computer screen, outside of Windows®. Download one of these pure Inner xSides now.

-▶ Marketplace
- ...▶ What is xSides?
-▶ What's Coming?
- ...▶ The Pixel Company
-▶ Help/FAQ

AllSides

DOWNLOAD NOW!



Lorum ipsum dolor sit amet consequat dolor sit amet consequat. Lorum ipsum dolor sit amet consequat dolor sit amet consequat. Lorum ipsum dolor sit amet consequat dolor sit amet consequat. Lorum ipsum dolor sit amet consequat dolor sit amet consequat. Lorum ipsum dolor sit amet consequat.

Category	Button	Button	Button	Button
----------	--------	--------	--------	--------

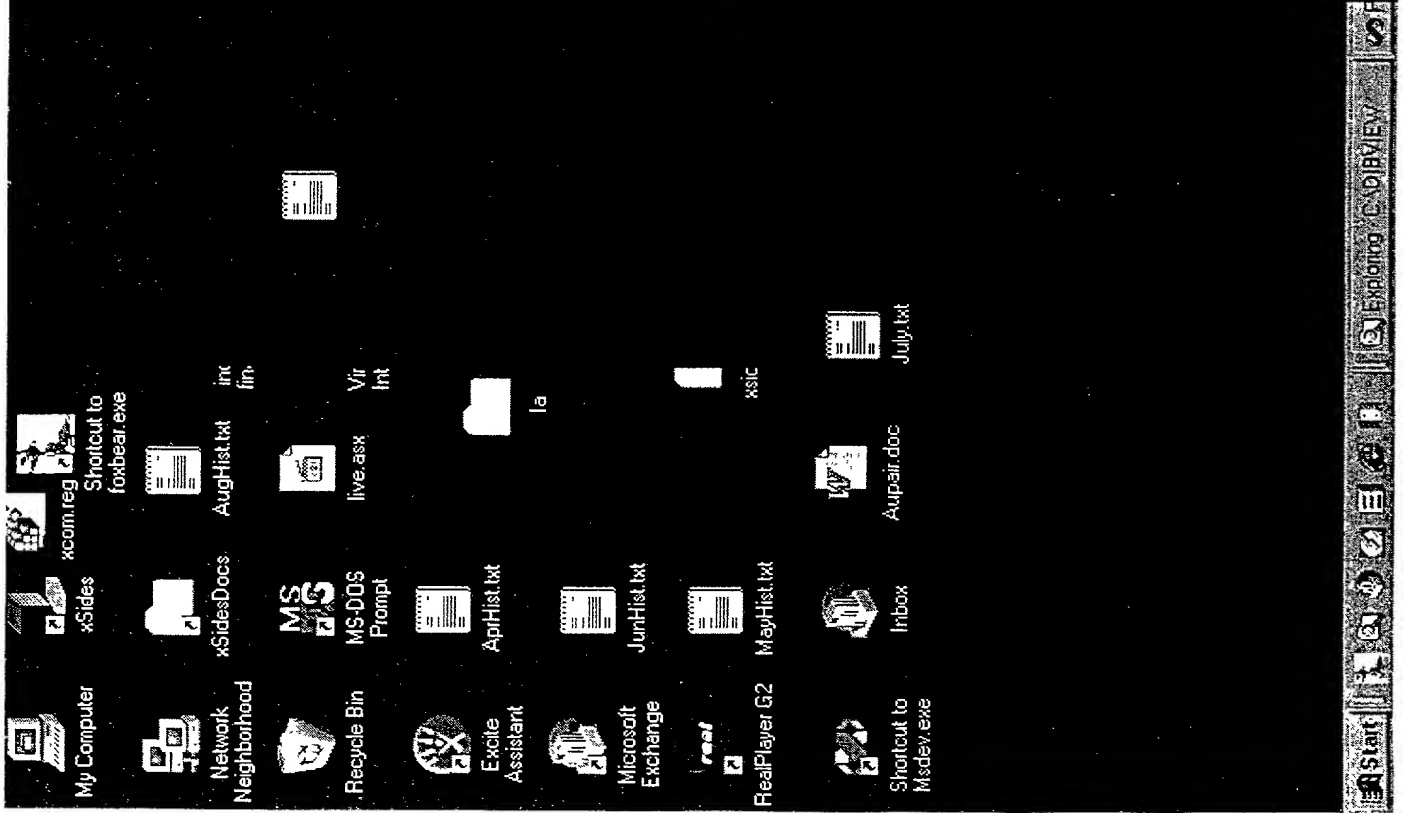
Lorum ipsum dolor sit amet consequat dolor sit amet consequat. Lorum ipsum dolor sit amet consequat dolor sit amet consequat. Lorum ipsum dolor sit amet consequat dolor sit amet consequat. Lorum ipsum dolor sit amet consequat.



System Requirements	Version Number	Compressed Size
Windows 95 or Windows NT 4.0	1.00.092	2,189,801 bytes

pixelcompany

00827" 8/542460



[illegible]

•
•
•
•
•
•
•
•
•
•
•

xSides Corporation

xSides™ Video Driver Extensions

• • • • •

Parameter	Value	Unit
Temperature	25.0	°C
Pressure	1.0	atm
Flow rate	1.0	L/min
Concentration	0.1	mol/L
pH	7.0	
Wavelength	254	nm
Scan rate	1.0	nm/min
Integration time	1.0	s
Resolution	0.1	nm
Detector	Photodiode array	
Injection volume	10	μL
Column	C18	
Mobile phase	Water/Acetonitrile	
Gradient	0-100% ACN in 10 min	
Flow rate	1.0	mL/min
Temperature	30.0	°C
Wavelength	254	nm
Scan rate	1.0	nm/min
Integration time	1.0	s
Resolution	0.1	nm
Detector	Photodiode array	
Injection volume	10	μL
Column	C18	
Mobile phase	Water/Acetonitrile	
Gradient	0-100% ACN in 10 min	
Flow rate	1.0	mL/min
Temperature	30.0	°C
Wavelength	254	nm
Scan rate	1.0	nm/min
Integration time	1.0	s
Resolution	0.1	nm
Detector	Photodiode array	
Injection volume	10	μL
Column	C18	
Mobile phase	Water/Acetonitrile	
Gradient	0-100% ACN in 10 min	
Flow rate	1.0	mL/min
Temperature	30.0	°C
Wavelength	254	nm
Scan rate	1.0	nm/min
Integration time	1.0	s
Resolution	0.1	nm
Detector	Photodiode array	
Injection volume	10	μL
Column	C18	
Mobile phase	Water/Acetonitrile	
Gradient	0-100% ACN in 10 min	
Flow rate	1.0	mL/min
Temperature	30.0	°C
Wavelength	254	nm
Scan rate	1.0	nm/min
Integration time	1.0	s
Resolution	0.1	nm
Detector	Photodiode array	
Injection volume	10	μL
Column	C18	
Mobile phase	Water/Acetonitrile	
Gradient	0-100% ACN in 10 min	
Flow rate	1.0	mL/min
Temperature	30.0	°C
Wavelength	254	nm
Scan rate	1.0	nm/min
Integration time	1.0	s
Resolution	0.1	nm
Detector	Photodiode array	
Injection volume	10	μL
Column	C18	
Mobile phase	Water/Acetonitrile	
Gradient	0-100% ACN in 10 min	
Flow rate	1.0	mL/min
Temperature	30.0	°C
Wavelength	254	nm
Scan rate	1.0	nm/min
Integration time	1.0	s
Resolution	0.1	nm
Detector	Photodiode array	
Injection volume	10	μL
Column	C18	
Mobile phase	Water/Acetonitrile	
Gradient	0-100% ACN in 10 min	
Flow rate	1.0	mL/min
Temperature	30.0	°C
Wavelength	254	nm
Scan rate	1.0	nm/min
Integration time	1.0	s
Resolution	0.1	nm
Detector	Photodiode array	
Injection volume	10	μL
Column	C18	
Mobile phase	Water/Acetonitrile	
Gradient	0-100% ACN in 10 min	
Flow rate	1.0	mL/min
Temperature	30.0	°C
Wavelength	254	nm
Scan rate	1.0	nm/min
Integration time	1.0	s
Resolution	0.1	nm
Detector	Photodiode array	
Injection volume	10	μL
Column	C18	
Mobile phase	Water/Acetonitrile	
Gradient	0-100% ACN in 10 min	
Flow rate	1.0	mL/min
Temperature	30.0	°C
Wavelength	254	nm
Scan rate	1.0	nm/min
Integration time	1.0	s
Resolution	0.1	nm
Detector	Photodiode array	
Injection volume	10	μL
Column	C18	
Mobile phase	Water/Acetonitrile	
Gradient	0-100% ACN in 10 min	
Flow rate	1.0	mL/min
Temperature	30.0	°C
Wavelength	254	nm
Scan rate	1.0	nm/min
Integration time	1.0	s
Resolution	0.1	nm
Detector	Photodiode array	
Injection volume	10	μL
Column	C18	
Mobile phase	Water/Acetonitrile	
Gradient	0-100% ACN in 10 min	
Flow rate	1.0	mL/min
Temperature	30.0	°C
Wavelength	254	nm
Scan rate	1.0	nm/min
Integration time	1.0	s
Resolution	0.1	

[illegible]

Overview

Introduction

This document describes the video driver extensions necessary to provide a generic interface between the xSides application and the Windows video driver. These extensions allow the display of xSides pixel bars in the video overscan region that exists at the edges of the video display, outside the Windows Desktop. The display of the xSides pixel bars, specifically the number of pixel bars or location of the pixel bars -- Top, Bottom, Right, or Left edge -- does not reduce the pixel resolution of the Windows desktop.

To implement the overscan regions, a set of video driver extensions are defined and implemented via driver escapes. The xSides application, which is a Win32 application running in user mode, creates, modifies, deletes and draws in the pixel bars by calling the GDI (Graphics Display Interface) Escape function, and passing it messages that the video driver understands. These messages are called the xSides Video Driver Extensions.

Support for the mouse cursor in the pixel bars is also defined to provide the xSides application the capability to define a cursor icon, and to position it in the pixel bars beyond the normal Windows Desktop.

Three levels of support are defined:

1. Level One supports only the bottom pixel bar.
2. Level Two supports both the bottom and the top pixel bars.
3. Level Three supports the bottom, top, left and right pixel bars.

Level One Support

Video drivers that provide xSides level one support allow the display of pixel bar (aka *Side*) at the bottom of the display, beyond the Windows desktop. The video driver can provide this capability by allocating video memory contiguous to the frame buffer, increasing the number of visible scanlines, and having the capability to BitBlt bitmaps to the allocated video memory.

Level Two Support

Video drivers that provide xSides level two support allow the display of pixel bars (aka *Sides*) at the top and the bottom of the display, above and below the Windows desktop respectively. The video driver can provide this capability by allocating enough video memory contiguous to the frame buffer for both pixel bars (aka *Sides*), increasing the number of visible scanlines sufficient for both pixel bars, the having capability to BitBlt bitmaps to either the top or bottom pixel bar, and offsetting the Windows Desktop downwards in the framebuffer so that the very top of the video framebuffer is taken up by the top pixel bar.

Level Three Support

Video drivers that provide xSides with level three support allow the display of pixel bars (aka *Sides*) at the left and right sides of the display, to the left and to the right, respectively, of the Windows desktop, in addition to the display of pixel bars at the top and bottom of the display. To provide this support, in addition to the capabilities described for level two support, the frame buffer could be organized so that the screen stride is larger than the Windows desktop horizontal byte count, to provide video memory for the left and right pixel bars, and the horizontal display resolution would need to be increased to make the pixel bars visible.

Mouse Cursor (Pointer) Support

Video drivers that provide xSides with mouse cursor support allow the definition and display of the cursor icon in the pixel bar screen regions, beyond the Windows desktop. To provide the support, the video driver must maintain a separate cursor icon cache for xSides, and use it when a request is made to display the mouse cursor in the xSides pixel bar(s). When a pixel bar is active along any edge of the desktop, the cursor icon should not be clipped at the desktop boundary, but rather should be extended onto the pixel bar until the cursor hotspot reaches the edge of the desktop. Once the cursor hotspot transitions out of the Windows Desktop, the normal cursor icon should be deactivated and made invisible.

Configuration Information

In order for the xSides application to size pixel bars appropriately, the video driver must return configuration information regarding the maximum amount of extra screen space that can be supported for each resolution. This is determined primarily by the amount of video memory and video bandwidth available. The xSides application will use the configuration information from the video driver, based upon the video adapter and monitor characteristics, and determine the optimum sizes for the pixel bars while minimizing refresh frequency degradation.

DirectDraw Support for xSides

There is interest in having the display driver support the assigning of DirectDraw surface handles to the pixel bars. The details are TBD.

Display Events

There is interest in having the display driver supporting the report of display events to the xSides application. These events would include:

- DirectDraw Full Screen start.
- DirectDraw Full Screen end.
- Screen Saver start.
- Screen Saver end.
- Dos Full Screen start.
- Dos Full Screen end.
- Message Mode (aka Blue Screen of Death) start.
- Message Mode (aka Blue Screen of Death) end.
- Display Resolution Change.
- Display Driver errors, especially in the xSides support code.

It's desired that the events be reported asynchronously, probably by callback functions. The details are TBD.

xSides Driver Escapes

The Video Driver must support an xSides Escape code (defined as ESC_xSIDES; the numerical value of the code is TBD), and report that it is being supported when queried with a QUERYYESCSUPPORT Escape code. In Windows NT, the GDI ExtEscape function will be used by the xSides application.

The Windows GDI escape functions are described in Appendix A.

xSides Escapes

All the xSides escapes use ESC_xSIDES for the nEscape parameter to the GDI function ExtEscape. The lpzInData parameter points to a structure that

contains the specific xSides message. The video driver returns output messages back to the xSides application in the buffer pointed to by the `lpzOutData` parameter. The following xSides messages are defined to support the display of the pixel bars:

- **IMSG_XSIDES_BEGIN** notifies the video driver that the xSides application has started executing and requires video driver support. The video driver returns the **OMSG_XSIDES_STATUS** message in response, with the appropriate status and error information.
- **IMSG_XSIDES_END** notifies the video driver that the xSides application is exiting and no longer requires video driver support. The video driver returns the **OMSG_XSIDES_STATUS** message in response, with the appropriate status and error information.
- **IMSG_XSIDES_QUERY_VIDEO_INFO** queries the video driver for information about the video hardware and driver. Among the information requested are the version of the xSides video driver extensions and the level of functionality that the video driver supports. The video driver returns the **OMSG_XSIDES_VIDEO_INFO** message.
- **IMSG_XSIDES_CREATE_PIXELBAR** commands the video driver to create the specified pixel bar -- bottom, top, left, or right. This includes allocating video memory for the pixel bar, reprogramming the video hardware to increase the vertical or horizontal pixel resolution, offsetting the Windows desktop (in the case of the top and left bars), and changing the screen stride (in the case of the left and right pixel bars.) The video driver returns the **OMSG_XSIDES_STATUS** message in response, with the appropriate status and error information.
- **IMSG_XSIDES_DELETE_PIXELBAR** commands the video driver to delete a previously created pixel bar. This may include deallocating video memory, reprogramming the video hardware, removing offsets from the display of the Windows desktop and changing the screen stride. The video driver returns the **OMSG_XSIDES_STATUS** message in response, with the appropriate status and error information.
- **IMSG_XSIDES_DISPLAY_PIXELBAR** commands the video driver to do a BitBlt of the specified bitmap to a previously created pixel bar at the specified position with the specified extents. The source bitmap is in the current screen format. For this message, the `lpzOutData` parameter points to the source bitmap.
- **IMSG_XSIDES_ACTIVATE_PIXELBAR** commands the video driver to make a pixel bar visible by increasing the number of pixels displayed on the display surface.

- **IMSG_XSIDES_DEACTIVATE_PIXELBAR** commands the video driver to do a.
- **IMSG_XSIDES_ENABLE_CURSOR** commands the video driver to display the mouse cursor icon in the pixel bar.
- **IMSG_XSIDES_DISABLE_CURSOR** commands the video driver to not display the mouse cursor icon in the pixel bar.
- **IMSG_XSIDES_SET_CURSOR** downloads the xSides mouse cursor icon to the video driver.
- **IMSG_XSIDES_MOVE_CURSOR** commands the video driver to move the xSides mouse cursor to a new location in the pixel bar.

xSides Escape Messages

Generic Message Structure

The xSides messages begin with a message code and size fields, followed by message specific data fields. The message code indicates the specific message; the size indicates the number of bytes in the message, including the code and size fields. In the figures below, Little Endian byte ordering (where the most significant byte of a multi-byte value has the highest address) is assumed, with the most significant byte at the left and the least significant byte at the right.

The generic xSides Escape message structure is as follows:

USHORT Code	USHORT Size
[Others -- Message Specific fields]	

Code: The message code, in bytes 2 and 3 of the message structure.

Size: The size of the message, including the Code and Size fields, in bytes 0 and 1 of the message structure.

Others: Message specific fields whose number, format and definition vary from message to message. Some messages may have none.

The structure would be defined in C as:

```
typedef struct _xSidesGenericEscapeMessage
{
    USHORT Size;

    USHORT Code;

    ULONG SpecificFields[1];
} xSidesGenericEscapeMessage;
```

Input Messages

The formats of the messages that the xSides application sends to the video driver are described below.

IMSG_XSIDES_BEGIN

IMSG_XSIDES_BEGIN is sent to the video driver to indicate that the xSides application has begun execution and requires video driver support.

USHORT Code	USHORT Size
-------------	-------------

Code: **IMSG_XSIDES_BEGIN**

Size: 4

IMSG XSIDES END

MSG_XSIDES_END is sent to the video driver to indicate that the xSides application is exiting, and no longer requires video driver support.

USHORT Code	USHORT Size
-------------	-------------

Code: **IMSG XSIDES END**

Size: 4

IMSG_XSIDES_QUERY_VIDEO_INFO

MSG_XSIDES_QUERY_VIDEO_INFO is sent to the video driver to query information about the video hardware and driver, including the version and level of functionality of xSides video driver extensions that the video

driver supports. The video driver returns the OMSG_XSIDES_VIDEO_INFO message.

USHORT Code	USHORT Size
-------------	-------------

Code: **IMSG_XSIDES_QUERY_VIDEO_INFO**

Size: 4

IMSG_XSIDES_CREATE_PIXELBAR

IMSG_XSIDES_CREATE_PIXELBAR is sent to the video driver to request the creation and display of a pixel bar on an edge of the display screen. The video driver returns the OMSG_XSIDES_STATUS message.

USHORT Code	USHORT Size
USHORT Edge	USHORT Extent
Flags	

Code: **IMSG_XSIDES_CREATE_PIXELBAR**

Size: 8

Edge: The edge of the screen to create the pixel bar on. May be XSIDES_EDGE_TOP, XSIDES_EDGE_BOTTOM, XSIDES_EDGE_LEFT, or XSIDES_EDGE_RIGHT, for the top, bottom, left and right edges, respectively. Only one pixel bar can be created per side; attempts to create further pixel bars on the same side will cause the video driver to return errors in the output message.

Extent: The number of scan lines high, or pixels wide that the pixel bar should be. Horizontal pixel bars (those on the top and bottom edges) span the entire current horizontal resolution of the display. Vertical pixel bars (those on the left and right edges) span the entire current vertical resolution of the display.

Flags: A field of flags that describe the properties of the pixel bar. Bit 0 is defined to be the active/inactive flag.

Bit 0: If 0, then the pixel bar is created in the inactive state, and is not visible, though memory for it is allocated from the video memory. If 1, then the pixel bar is created in the active state and space is created for it on the display surface.

IMSG_XSIDES_DELETE_PIXELBAR

IMSG_XSIDES_DELETE_PIXELBAR is sent to the video driver to delete a pixel bar previously created by an **IMSG_XSIDES_CREATE_PIXELBAR** message. The video driver returns the **OMSG_XSIDES_STATUS** message.

USHORT Code	USHORT Size
USHORT Edge	USHORT Reserved

Code: **IMSG_XSIDES_DELETE_PIXELBAR**

Size: 8

Edge: The edge of the screen that the pixel bar is on. May be **XSIDES_EDGE_TOP**, **XSIDES_EDGE_BOTTOM**, **XSIDES_EDGE_LEFT**, or **XSIDES_EDGE_RIGHT**, for the top, bottom, left and right edges, respectively. Attempts to delete a pixel bar that doesn't exist will cause the video driver to return error reports in the output message.

IMSG_XSIDES_ACTIVATE_PIXELBAR

IMSG_XSIDES_ACTIVATE_PIXELBAR is sent to the video driver to enable the display of a pixel bar previously created by an **IMSG_XSIDES_CREATE_PIXELBAR** message. The video driver returns the **OMSG_XSIDES_STATUS** message.

USHORT Code	USHORT Size
USHORT Edge	USHORT Reserved

Code: **IMSG_XSIDES_ACIVATE_PIXELBAR**

Size: 8

Edge: The edge of the screen that the pixel bar is on. May be **XSIDES_EDGE_TOP**, **XSIDES_EDGE_BOTTOM**, **XSIDES_EDGE_LEFT**, or **XSIDES_EDGE_RIGHT**, for the top, bottom, left and right edges, respectively. Attempts to activate a pixel bar that doesn't exist, or is already being displayed, will cause the video driver to return error reports in the output message.

IMSG_XSIDES_DEACTIVATE_PIXELBAR

MSG_XSIDES_DEACTIVATE_PIXELBAR is sent to the video driver to disable the display of a pixel bar previously created by an **MSG_XSIDES_CREATE_PIXELBAR** message. The video driver returns the **MSG_XSIDES_STATUS** message.

USHORT Code	USHORT Size
USHORT Edge	USHORT Reserved

Code: **IMSG_XSIDES_DEACIVATE_PIXELBAR**

Size: 8

Edge: The edge of the screen that the pixel bar is on. May be `XSIDES_EDGE_TOP`, `XSIDES_EDGE_BOTTOM`, `XSIDES_EDGE_LEFT`, or `XSIDES_EDGE_RIGHT`, for the top, bottom, left and right edges, respectively. Attempts to deactivate a pixel bar that doesn't exist, or is already being inactive, will cause the video driver to return error reports in the output message.

IMSG_XSIDES_DISPLAY_PIXELBAR

MSG_XSIDES_DISPLAY_PIXELBAR is sent to the video driver to display bitmaps on a pixel bar previously created by an **MSG_XSIDES_CREATE_PIXELBAR** message. The video driver uses the **SRCCOPY (0xCC)** ROP to **BitBlt** the specified bitmap onto the pixel bar. The source bitmap address is specified in the **lpSrcOutData** field of the **Escape** function. The video driver returns the **MSG_XSIDES_STATUS** message.

USHORT Code	USHORT Size
USHORT Edge	USHORT SrcStride
USHORT DestX	USHORT DestY
USHORT SrcX	USHORT SrcY
USHORT ExtX	USHORT ExtY

Code: **IMSG XSIDES DISPLAY_PIXELBAR**

Size: 20

Edge: The edge of the screen that the pixel bar is on. May be `XSIDES_EDGE_TOP`, `XSIDES_EDGE_BOTTOM`, `XSIDES_EDGE_LEFT`, or `XSIDES_EDGE_RIGHT`, for the top, bottom, left and right edges, respectively. Attempts to

display to a pixel bar that doesn't exist will cause the video driver to return an error.

SrcStride: The stride of the source bitmap. I.e. the number of bytes from the first pixel of a row to the first pixel of the next row in the bitmap.

DestX: The horizontal (x) location in the destination pixel bar, relative to the pixel bar origin (the pixel bar origin is at the upper left corner of the pixel bar, with x values increasing to the right and y values increasing towards the bottom.) where the source bitmap is to be written.

DestY: The vertical (y) location in the destination pixel bar, relative to the pixel bar origin (the pixel bar origin is at the upper left corner of the pixel bar, with x values increasing to the right and y values increasing towards the bottom.) where the source bitmap is to be written.

SrcX: The horizontal (x) location in the source bitmap, relative to the bitmap origin (the bitmap origin is at the upper left corner of the bitmap, with x values increasing to the right and y values increasing towards the bottom.) where the SRCCOPY BitBlt starts transferring data from.

SrcY: The vertical (y) location in the source bitmap, relative to the bitmap origin (the bitmap origin is at the upper left corner of the bitmap, with x values increasing to the right and y values increasing towards the bottom.) where the SRCCOPY BitBlt starts transferring data from.

MSG_XSIDES_ENABLE_CURSOR

MSG_XSIDES_ENABLE_CURSOR is sent to the video driver to enable the display of a mouse cursor icon a pixel bar created by an **MSG_XSIDES_CREATE_PIXELBAR** message. The video driver returns the **MSG_XSIDES_STATUS** message.

USHORT Code	USHORT Size
USHORT Edge	USHORT Reserved

Code: **MSG_XSIDES_ENABLE_CURSOR**

Size: 8

Edge: The edge of the screen that the pixel bar is on. May be **XSIDES_EDGE_TOP**, **XSIDES_EDGE_BOTTOM**, **XSIDES_EDGE_LEFT**, or **XSIDES_EDGE_RIGHT**, for the top, bottom, left and right edges, respectively. Attempts to enable cursor display in pixel bar that doesn't exist, or where it is already enabled, will cause the video driver to return error reports in the output message.

IMSG_XSIDES_DISABLE_CURSOR

IMSG_XSIDES_DISABLE_CURSOR is sent to the video driver to enable the display of a mouse cursor icon a pixel bar created by an **IMSG_XSIDES_CREATE_PIXELBAR** message. The video driver returns the **OMSG_XSIDES_STATUS** message.

USHORT Code	USHORT Size
USHORT Edge	USHORT Reserved

Code: **IMSG_XSIDES_DISABLE_CURSOR**

Size: 8

Edge: The edge of the screen that the pixel bar is on. May be **XSIDES_EDGE_TOP**, **XSIDES_EDGE_BOTTOM**, **XSIDES_EDGE_LEFT**, or **XSIDES_EDGE_RIGHT**, for the top, bottom, left and right edges, respectively.

Note: Attempts to disable cursor display in pixel bar that doesn't exist, or where it is already disabled, will cause the video driver to return error reports in the output message.

IMSG_XSIDES_SET_CURSOR

IMSG_XSIDES_DISABLE_CURSOR is sent to the video driver to enable the display of a mouse cursor icon a pixel bar created by an **IMSG_XSIDES_CREATE_PIXELBAR** message. The video driver returns the **OMSG_XSIDES_STATUS** message. The cursor bitmap address is specified in the **lpszOutData** field of the **Escape** function.

USHORT Code	USHORT Size
USHORT Edge	USHORT Reserved

Code: **IMSG_XSIDES_DISABLE_CURSOR**

Size: 8

Edge: The edge of the screen that the pixel bar is on. May be **XSIDES_EDGE_TOP**, **XSIDES_EDGE_BOTTOM**, **XSIDES_EDGE_LEFT**, or **XSIDES_EDGE_RIGHT**, for the top, bottom, left and right edges, respectively.

IMSG_XSIDES_MOVE_CURSOR

IMSG_XSIDES_MOVE_CURSOR is sent to the video driver to position the cursor in a pixel bar created by an **IMSG_XSIDES_CREATE_PIXELBAR** message. The video driver returns the **OMSG_XSIDES_STATUS** message.

USHORT Code	USHORT Size
USHORT Edge	USHORT Reserved
USHORT DestX	USHORT DestY

Code: **IMSG_XSIDES_DISPLAY_PIXELBAR**

Size: 12

Edge: The edge of the screen that the pixel bar is on. May be **XSIDES_EDGE_TOP**, **XSIDES_EDGE_BOTTOM**, **XSIDES_EDGE_LEFT**, or **XSIDES_EDGE_RIGHT**, for the top, bottom, left and right edges, respectively. **DestX**: The horizontal (x) location in the destination pixel bar, relative to the pixel bar origin (the pixel bar origin is at the upper left corner of the pixel bar, with x values increasing to the right and y values increasing towards the bottom.) where the mouse cursor hotspot is to be positioned.

DestY: The vertical (y) location in the destination pixel bar, relative to the pixel bar origin (the pixel bar origin is at the upper left corner of the pixel bar, with x values increasing to the right and y values increasing towards the bottom.) where the mouse cursor hotspot it to be positioned.

Note: Attempts to position the cursor in a pixel bar that doesn't exist, or where the cursor isn't enabled, will cause the video driver to return an error.

Output Messages

OMSG_XSIDES_VIDEO_INFO

OMSG_XSIDES_VIDEO_INFO is returned by the video driver to the xSides Applications in response to an **IMSG_XSIDES_QUERY_VIDEO_INFO** message, and describes the version and level of xSides support provided as well as the video driver version and pertinent data about the video hardware.

USHORT Code	USHORT Size
USHORT	USHORT xSidesLevel

xSidesVersion	
VideoDriverVersion	
VideoHardware	
VideoConfigurationData	

Code: **OMSG_XSIDES_VIDEO_INFO**

Size: **TBD**

xSidesVersion: The version of xSides video driver extensions that the video driver supports.

xSidesLevel: The level of xSides functionality that the video driver supports. May be 0 (no support), 1 (bottom pixel bar only), 2 (bottom and top pixel bars) or 3 (bottom, top, left and right pixel bars.).

VideoDriverVersion: The name and version (release number) of the video driver and video miniport. This should be a much larger field, or set of fields, including character strings for file names.
Note: the format is TBD.

VideoHardware: The names of the video hardware, including the board and chip, their manufacturers, versions and revisions, characteristics (e.g. amount of video RAM, the video bus interface, etc.). **Note: the format is TBD.**

VideoConfigurationData: The enumeration of the display resolutions supported, and the xSides resources available for each resolution, including specification of the number of extra pixels available and the resultant refresh frequency degradation.
Note: the format is TBD.

OMSG_XSIDES_STATUS

OMSG_XSIDES_STATUS is returned by the video driver to the xSides application in response to various xSides input messages to report status and errors.

USHORT Code	USHORT Size
USHORT RequestingCode	USHORT RequestingSize
ULONG ErrorCode	

ErrorCode: The status and error code being returned. This is TBD; more fields including test strings may be required.

15

Specifies the number of bytes of data pointed to by the *lpvInData* parameter.

Pointer to the input structure required for the specified .

Pointer to the structure that receives output from this. This parameter should be NULL if no data is returned.

If the function succeeds, the return value is greater than zero, except with the **QUERYYESCSUPPORT** printer, which checks for implementation only. If the is not implemented, the return value is zero.

GDI ExtEscape Function

The **ExtEscape** function allows applications to access capabilities of a particular device that are not available through GDI.

Parameters

Handle to the device context.

Specifies the escape function to be performed.

Specifies the number of bytes of data pointed to by the *lpszInData* parameter.

Pointer to the input structure required for the specified escape.

Specifies the number of bytes of data pointed to by the *lpSzOutData* parameter.

Pointer to the structure that receives output from this escape. This parameter must not be NULL if **ExtEscape** is called as a query function. If no data is to be returned in this structure set *cbOutput* to 0.

The return value specifies the outcome of the function. It is greater than zero if the function is successful, except for the `QUERYESCSUPPORT` printer escape, which checks for implementation only. The return value is zero if the escape is not implemented. A return value less than zero indicates an error.

DrvEscape

DrvEscape is used for retrieving information from a device that is not available in a device-independent device driver interface. The particular query depends on the value of the *iEsc* parameter. Drawing on the device is not allowed in this function.

DrvDrawEscape is to be used for specialized drawing support.

```

ULONG DrvEscape(
    IN SURFOBJ*pso,
    IN ULONG iEsc,
    IN ULONG cjIn,
    IN PVOID*pvin,
    IN ULONG cjOut,
    OUT PVOID*pvOut

```



pso

Points to a SURFOBJ structure that describes the surface to which the call is directed.

iEsc

Specifies a query. The meaning of the other parameters depends on this value. ESC_QUERYESCSUPPORT is the only predefined value; it queries whether the driver supports a particular escape function. In this case, *pvIn* points to an escape function number; *cjOut* and *pvOut* are ignored. If the specified function is supported, the return value is nonzero.

cjIn

Specifies the size, in bytes, of the buffer pointed to by *pvIn*.

pvIn

Points to the input data for the call. The format of the input data depends on the query specified by the *iEsc* parameter.

cjOut

Specifies the size, in bytes, of the buffer pointed to by *pvOut*.

pvOut

Points to the output buffer. The format of the output data depends on the query specified by the *iEsc* parameter.

Return Value

The return value is dependent on the query specified by the *iEsc* parameter. If the function specified in the query is not supported, the return value is zero.

00000000 "ESC" 00000000

19

Appendix F

4.

42

Variable	Mean	SD	Min	Max
Age	34.5	10.2	18	55
Gender	1.0	0.0	0	1
Marital status	1.5	0.5	0	2
Education	12.5	1.5	9	16
Income	1.2	0.3	0.5	2.0
Occupation	1.0	0.0	0	1
Health status	1.0	0.0	0	1
Smoking status	1.0	0.0	0	1
Alcohol consumption	1.0	0.0	0	1
Exercise frequency	1.0	0.0	0	1
Stress level	1.0	0.0	0	1
Depression score	1.0	0.0	0	1
Life satisfaction	1.0	0.0	0	1
Quality of life	1.0	0.0	0	1
Health-related quality of life	1.0	0.0	0	1
Physical health	1.0	0.0	0	1
Mental health	1.0	0.0	0	1
Social health	1.0	0.0	0	1
Environmental health	1.0	0.0	0	1
Overall health	1.0	0.0	0	1
Health status	1.0	0.0	0	1
Smoking status	1.0	0.0	0	1
Alcohol consumption	1.0	0.0	0	1
Exercise frequency	1.0	0.0	0	1
Stress level	1.0	0.0	0	1
Depression score	1.0	0.0	0	1
Life satisfaction	1.0	0.0	0	1
Quality of life	1.0	0.0	0	1
Health-related quality of life	1.0	0.0	0	1
Physical health	1.0	0.0	0	1
Mental health	1.0	0.0	0	1
Social health	1.0	0.0	0	1
Environmental health	1.0	0.0	0	1
Overall health	1.0	0.0	0	1

The Application Programmer's Interface to the Pixel Bar

Table of Contents

TABLE OF CONTENTS.....	2
INTRODUCTION.....	3
PIXEL BAR API FUNCTIONS.....	5
PIXELBAR_INIT.....	5
PIXELBAR_UNINIT.....	6
PIXELBAR_CREATE.....	6
PIXELBAR_CREATEMR.....	9
PIXELBAR_MODIFY.....	11
PIXELBAR_DELETE.....	12
PIXELBAR_ACTIVATE.....	13
PIXELBAR_DEACTIVATE.....	13
PIXELBAR_ENUMERATE.....	14
PIXELBAR_QUERY.....	15
PIXELBAR_SETMOUSEEVENT.....	16
PIXELBAR_CLEARMOUSEEVENT.....	18
PIXELBAR_ENUMERATEMOUSEEVENTS.....	19
PIXELBAR_QUERYMOUSEEVENT.....	20
PIXELBAR_ENABLEMOUSE.....	21
PIXELBAR_DISABLEMOUSE.....	22
PIXELBAR_QUERYDISPLAYINFO.....	22
PIXELBAR_DISPLAYCOLOR.....	23
PIXELBAR_DISPLAYBITMAP.....	25
PIXELBAR_SETDISPLAYEVENT.....	26
PIXELBAR_CLEARDISPLAYEVENT.....	29
PIXELBAR_ENUMERATEDISPLAYEVENTS.....	31
PIXELBAR_QUERYSUPPORT.....	32
PIXELBAR_ENUMERATEFILEINFORMATION.....	34
DATA DEFINITIONS.....	35
INTEGER TYPES.....	35
MACROS.....	36
Return Codes:.....	36
xSides Modes:.....	37
Status Flags: // Why aren't these called *_Masks?.....	37
Display Edges: // Change from bitmasks to ordinals.	37
Display Edge Masks:.....	38
Mouse Events:.....	38
Display Events:.....	38
xSides Driver Error.....	39
XSD_FILE_TYPE:.....	39
STRUCTURE TYPES.....	40
PBConfiguration.....	40
PBDisplayInfo.....	41
PBPixelbarState.....	43
CURSOR_POS.....	44
PBMouseEvent.....	45
PBMouseEventInfo.....	46
PBDisplayEvent.....	48
PBPixelbarSize.....	49

<i>PBSize</i>	51
<i>PBFileInfo</i>	51
INDEX	53
PIXEL BAR HEADER FILE	54

Introduction

The Pixel Bar API is a set of C style functions that enable the xSides application to define pixel bars and display device dependent bitmaps in the pixel bars. The API includes functions to initialize the interface, create, modify, query and delete pixel bars, and to display colors or bitmaps in the pixel bars. For Win32 implementations, the API functions exist in a dynamically linked library (DLL).

The API functions include:

- **Pixelbar_Init** initializes and configures the API and the Pixel Bar software. The API functions are enabled.
- **Pixelbar_Uninit** restores the API and the Pixel Bar software to default undefined state. Other than Pixelbar_Init, all API functions are disabled.
- **Pixelbar_Create** creates a pixel bar and specifies its characteristics -- location, size, and whether it's visible or not. Also specifies the mode to use to acquire the screen area.
- **Pixelbar_CreateMR** creates a pixel bar and specifies its characteristics -- location, size, and whether it's visible or not for each supported display resolution. The mode used to acquire the screen area is also specified. This function differs from Pixelbar_Create in that when the display resolution changes, the xSides Space is automatically resized without further input from the xSides application.
- **Pixelbar_Modify** modifies the state of an existing pixel bar.
- **Pixelbar_Delete** deletes an existing pixel bar. The screen area is reclaimed, and the pixel bar is no longer displayed.
- **Pixelbar_Activate** makes a pixel bar active and visible on the screen.
- **Pixelbar_Deactivate** makes a pixel bar inactive and no longer visible on the screen. The screen area formerly occupied by the pixel bar is reclaimed.
- **Pixelbar_Query** returns the current state of the specified pixel bar, including location, size, visibility, etc.
- **Pixelbar_Enumerate** returns the number of pixel bars and their state.
- **Pixelbar_SetMouseEvent** sets a mouse event that will be reported to the application via a callback function.

- **Pixelbar_ClearMouseEvent** deletes the specified mouse event. The application will no longer receive reports for the mouse event.
- **Pixelbar_EnumerateMouseEvents** returns the list of current mouse events that the pixel bar software reports.
- **Pixelbar_QueryMouseEvent** returns the information of a specific mouse event the pixel bar software services.
- **Pixelbar_EnableMouse** enables the display of the mouse cursor icon in a pixel bar.
- **Pixelbar_DisableMouse** disables the display of the mouse cursor icon in a pixel bar.
- **Pixelbar_QueryDisplayInfo** returns the current physical screen resolution and other display information (e.g. refresh rate). The physical screen resolution doesn't include the screen areas occupied by pixel bars in overscan mode.
- **Pixelbar_DisplayColor** displays a color across the entire specified pixel bar. Used for testing.
- **Pixelbar_DisplayBitmap** displays a device dependent bitmap (i.e. a bitmap in the current screen format) at the specified location in the specified pixel bar.
- **Pixelbar_SetDisplayEvent** sets a display event that will be reported back to the application via a callback function.
- **Pixelbar_ClearDisplayEvent** deletes the specified display event.
- **Pixelbar_EnumerateDisplayEvents** returns the list of display events that are currently set.
- **Pixelbar_QuerySupport** returns the capabilities of the virtual xSides device.
- **Pixelbar_EnumerateFileInformation** returns the list of xSides driver files, their locations, names and versions.

A number of data structure types are used by the Pixel Bar API. They include:

- **PBConfiguration** describes the configuration of the pixel bar software for the current run.
- **PBPixelBarState** describes the current state of a pixel bar.
- **PBDisplayState** describes the current state of a display.
- **CURSOR_POS** contains the x and y coordinates of the cursor.

- **PBMouseEvent** contains the information that is passed back to the application's mouse event callback function in response to a mouse event.
- **PBMouseEventInfo** describes the current state of a mouse event set.
- **PBDisplayEvent** contains the information that is passed back to the application in response to a display event.

Pixel Bar API Functions

Pixelbar_Init

UINT32 Pixelbar_Init (PBConfiguration *lpPBConfiguration)

Initializes the pixel bar software and configures it using the data in the PBConfiguration parameter.

- Returns one of these values:

PB_SUCCESS

Function was successful.

PB_ERROR

An unrecoverable error occurred.

Parameters

LpPBConfiguration

Pointer to the configuration data structure. The pixel bar software is configured for the current session by the values in this data structure.

Notes

None.

Pixelbar_Uninit

UINT32 Pixelbar_Uninit(void)

Uninitializes the pixel bar software restores it to the uninitialized state. All resources allocated and used by the pixel bar software during an xSides session is freed.

- Returns one of these values:

PB_SUCCESS

Function was successful.

PB_ERROR

An unrecoverable error occurred.

Parameters

This function has no parameters.

Notes

None.

Pixelbar_Create

UINT32 Pixelbar_Create (UINT32 *lpPixelbarId, UINT32 xSidesMode, UINT32 Flags, UINT32 DisplayId, UINT32 Edge, INT32 OriginX, INT32 OriginY, UINT32 Width, UINT32 Height)

Pixelbar_Create() creates a pixel bar and returns the id. Memory is allocated for the pixel bar and, if created with the active flag, screen space is created on the display surface.

- Returns one of these values:

PB_SUCCESS	Function was successful.
PB_ERROR	An unrecoverable error occurred.

Parameters

lpPixelbarId

A pointer to the pixel bar id returned by Pixelbar_Create().

xSidesMode

The xSides mode that the pixel bar is to operate in. May be one of the following:

- MODE_SHARE Share mode -- shares the current display resolution with the desktop by decreasing the desktop size.
- MODE_STEPUP Step Up mode -- increases the physical resolution of the display, but keep the desktop at the current resolution and use the difference for displaying the pixel bar.
- MODE_STEPDOWN Step Down mode -- maintains the physical resolution of the display, but force the desktop to the next smaller defined resolution and use the difference for displaying the pixel bar.
- MODE_OVERSCAN Overscan mode -- reprograms the video hardware to provide extra scanlines at the bottom of the screen to display the pixel bar. This mode currently is valid only for pixel bars at the bottom or top edge of the display.
- MODE_AUTO Automatic mode -- the pixel bar software will attempt to use Overscan mode first, and if that's not possible, then use Share mode,

Flags

The options are specified by the following bitfields:

- $$DisplayId$$

Edge

- PB_TOP_EDGE The top edge.
- PB_BOTTOM_EDGE The bottom edge.
- PB_RIGHT_EDGE The right edge. This edge is not valid if the pixel bar is in Overscan mode.
- PB_LEFT_EDGE The left edge. This edge is not valid if the pixel bar is in Overscan mode.

OriginX specifies the x position on the display surface, in units of pixels with respect to the Desktop Origin, where the origin of the pixel bar (which is the upper left corner of the pixel bar) is located. The pixel bar origin must be consistent with its screen edge parameter; otherwise an error will be returned. This is a signed value and may be a negative value (which it will be for non-zero width pixel bars on the left edge of the screen.)

OriginY

OriginY specifies the y position on the display surface, in units of pixels with respect to the Desktop Origin, where the origin of the pixel bar (which is the upper left corner of the pixel bar) is located. The pixel bar origin must be consistent with its screen edge parameter; otherwise an error will be returned. This is a signed value and may be a negative value (which it will be for non-zero height pixel bars on the top edge of the screen.)

Width

Width specifies the desired width of the pixel bar, in number of pixels. It may not be larger than the total physical horizontal resolution of the display, which include the desktop and the pixel bars.

Height

Height specifies the desired height of the pixel bar, in number of pixels. It may not be larger than the total physical vertical resolution of the display, which include the desktop and the pixel bars.

Notes

The interaction of pixel bars running in different xSides modes is currently undefined, and therefore not supported. Hence, in a session, all pixel bars must be in the same xSides mode.

Pixelbar_CreateMR

UINT32 Pixelbar_CreateMR(UINT32 *lpPixelbarId, UINT32 xSidesMode, UINT32 Flags, UINT32 DisplayId, UINT32 Edge, PBPixelbarSize *lpPixelbarSize, UINT32 xSidesResolutionCount)

Pixelbar_CreateMR() creates a pixel bar and returns the id. This functions replaces *Pixelbar_Create()* and allows pixel bar sizes for various resolutions to be specified. This function differs from *Pixelbar_Create* in that when the display resolution changes, the xSides Space is automatically resized without further input from the xSides application.

- Returns one of these values:

PB_SUCCESS	Function was successful.
PB_ERROR	An unrecoverable error occurred.

Parameters*lpPixelbarId*

A pointer to the pixel bar id returned by *Pixelbar_CreateMR()*.

xSidesMode

The xSides mode that the pixel bar is to operate in. May be one of the following:

- **MODE_SHARE** Share mode -- shares the current display resolution with the desktop by decreasing the desktop size.
- **MODE_STEPUP** Step Up mode -- increases the physical resolution of the display, but keep the desktop at the current resolution and use the difference for displaying the pixel bar.
- **MODE_STEPDOWN** Step Down mode -- maintains the physical resolution of the display, but force the desktop to the next smaller defined resolution and use the difference for displaying the pixel bar.
- **MODE_OVERSCAN** Overscan mode -- reprograms the video hardware to provide extra scanlines at the bottom of the screen to display the pixel bar. This mode currently is valid only for pixel bars at the bottom or top edge of the display.
- **MODE_AUTO** Automatic mode -- the pixel bar software will attempt to use Overscan mode first, and if that's not possible, then use Share mode,

Flags

Flags specifies the options and features of the Pixelbar_Create() function and of the pixel bar that is created.

The options are specified by the following bitfields:

- **PB_FLAGS_ACTIVE:** Can be TRUE (1) or FALSE (0), which specify, respectively, whether the pixel bar is active or inactive upon creation. Active pixel bars are visible while inactive pixel bars are not visible, but may still be operated upon and updated.
- **PB_FLAGS_EXACT_SIZE:** Can be either TRUE (1) or FALSE (0), which specify respectively whether the pixel bar should be created with the exact size specified, returning an error if it's not possible, or with the closest size possible if there are not sufficient resources to provide the specified size. If the application creates the pixel bar with the PB_STATUS_EXACT_SIZE set to FALSE, it should query for the actual size created with Pixelbar_Query() function.
- **PB_FLAGS_INVALID:** What is this?
- **PB_FLAGS_VALID_RES:** What is this?
- **PB_FLAGS_MOUSE_ENABLED:** What is this?
- **PB_FLAGS_DISPLAYED:** What is this?

DisplayId

DisplayId specifies the display surface (aka monitor, screen) that the pixel bar will be displayed on. This feature supports multiple monitor systems. Currently, only DisplayId of 0 is supported.

Edge

Edge specifies which edge of the display surface the pixel bar will be displayed on. May be one of the following:

PB_TOP_EDGE	The top edge.
PB_BOTTOM_EDGE	The bottom edge.
PB_RIGHT_EDGE	The right edge. This edge is not valid if the pixel bar is in Overscan mode.
PB_LEFT_EDGE	The left edge. This edge is not valid if the pixel bar is in Overscan mode.

lpPixelbarSize

lpPixelbarState is a pointer to an array of pixel bar size and origin data for various screen resolutions. The number of entries in the array is specified by xSidesResolutionCount.

xSidesResolutionCount

xSidesResolutionCount indicates the number of xSides resolutions that are listed by lpPixelbarSize.

Notes

The interaction of pixel bars running in different xSides modes is currently undefined, and therefore not supported. Hence, in a session, all pixel bars must be in the same xSides mode.

Pixelbar_Modify

UINT32 Pixelbar_Modify (UINT32 PixelbarId, UINT32 Flags, INT32 OriginX, INT32 OriginY, UINT32 Width, UINT32 Height)

Pixelbar_Modify() modifies the size and position of an existing pixel bar. The xSides mode and the edge are not affected.

- Returns one of these values:

Function was successful.

An unrecoverable error occurred.

PixelbarId

Status

The options are specified by the following bitfields:

PB_STATUS_EXACT_SIZE: Can be either TRUE (1) or FALSE (0), which specify respectively whether the pixel bar should be created with the exact size specified, returning an error if it's not possible, or with the closest size possible if there are not sufficient resources to provide the specified size. If the application creates the pixel bar with the PB_STATUS_EXACT_SIZE set to FALSE, it should query for the actual size created with `Pixelbar_Query()` function.

OriginX specifies the x position on the display surface, in units of pixels with respect to the Desktop Origin, where the origin of the pixel bar (which is the upper left corner of the pixel bar) is located. The pixel bar origin must be consistent with its screen edge parameter; otherwise an error will be returned. This is a signed value and may be a negative value (which it will be for non-zero width pixel bars on the left edge of the screen.)

OriginY specifies the y position on the display surface, in units of pixels with respect to the Desktop Origin, where the origin of the pixel bar (which is the upper left corner of the pixel bar) is located. The pixel bar origin must be consistent with its screen edge parameter; otherwise an error will be returned. This is a signed value and may be a negative value (which it will be for non-zero height pixel bars on the top edge of the screen.)

Width specifies the desired width of the pixel bar, in number of pixels. It may not be larger than the total physical horizontal resolution of the display, which include the desktop and the pixel bars.

The Pixel Bar API

Notes

Pixelbar_Delete

Pixelbar_Delete() deletes an existing pixel bar and frees its resources.

- | | |
|------------|----------------------------------|
| PB_SUCCESS | Function was successful. |
| PB_ERROR | An unrecoverable error occurred. |

PixelbarId

Notes

Pixelbar_Activate

Pixelbar_Activate() activates an existing pixel bar and makes it visible. If the pixel bar uses **MODE_SHARE**, the desktop is reduced and space on the display surface is created for the pixel bar. . If the pixel bar uses **MODE_OVERSCAN**, the video hardware is reprogrammed to create the scan lines used by the pixel bar

- | | |
|------------|----------------------------------|
| PB_SUCCESS | Function was successful. |
| PB_ERROR | An unrecoverable error occurred. |

Parameters

PixelbarId

The id of the pixel bar.

Notes

None.

Pixelbar_Deactivate

UINT32 Pixelbar_Deactivate (UINT32 PixelbarId)

Pixelbar_Deactivate() deactivates an existing pixel bar and makes it no longer visible. If the pixel bar uses MODE_SHARE, then the screen space is returned to the desktop. If the pixel bar uses MODE_OVERSCAN, the video hardware is reprogrammed to remove the scan lines used by the pixel bar.

- Returns one of these values:

PB_SUCCESS	Function was successful.
PB_ERROR	An unrecoverable error occurred.

Parameters

PixelbarId

The id of the pixel bar.

Notes

The behavior Pixelbar_Deactivate() if the pixel bar uses MODE_STEPUP or MODE_STEPDOWN is currently neither defined nor supported..

Pixelbar_Enumerate

UINT32 Pixelbar_Enumerate (UINT32 *lpPixelbarCount, UINT32 **lpPixelbarIdTable, UINT32 *lpPixelbarsReturned, UINT32 *lpszData, UINT32 szOutputBuffer)

Pixelbar_Enumerate() returns the count of existing pixel bars and their ids. The application can then get the characteristics of the individual pixel bars by calling Pixelbar_Query().

- Returns one of these values:

PB_SUCCESS	Function was successful.
PB_ERROR	An unrecoverable error occurred.

The provided output buffer is too small to contain all the requested information. The information returned in the output buffer is not guaranteed to be coherent or valid. The function should be called again with an output buffer of the appropriate size.

lpPixelbarCount

lpPixelbarIdTable

lpPixelbarsReturned

lpszData

szOutputBuffer

Notes

Pixelbar_Query

`Pixelbar_Query()` returns the state and characteristics of the specified pixel bar, including whether it is active, its width and height, etc.

- ## The Pixel Bar API

Parameters

The id of the pixel bar.

The address of a `PBPixelbarState` structure that the pixel bar state is returned in.

The address of the size of the requested data. If the value is larger than `szOutputBuffer`, then the provided buffer is too small, and the error code `PB_BUFFER_TOO_SMALL` is returned by the function.

The size of the output buffer pointed to by `lpPBState`, in bytes. This will allow the Pixel Bar software to refrain from overflowing the buffer.

None.

UINT32 Pixelbar_SetMouseEvent (UINT32 PixelbarId, UINT32 *lpMouseEventId, UINT32 MouseEvent, UINT32 (*fpCallback)(PBMouseEvent *lpParam))

The Pixel Bar API

specify the same mouse events (e.g. PB_MOUSE_MOVEMENT), all requests are serviced in the TBD [*possibly reverse*] order that they were requested. The notification is performed via function callbacks.

- Returns one of these values:

PB_SUCCESS	Function was successful.
PB_ERROR	An unrecoverable error occurred.

Parameters

Pixelbar_Id

The id of the pixel bar.

lpMouseEventId

The address of a UINT32 that in which the Pixel Bar software returns the Mouse Event Id.

MouseEvent

The mouse event that triggers the callback function. It may be one or more of the following:

PB_MOUSE_LBUTTON_DOWN	Trigger when the cursor is in the pixel bar and the left mouse button is depressed.
PB_MOUSE_LBUTTON_UP	Trigger when the cursor is in the pixel bar and the left mouse button is released.
PB_MOUSE_RBUTTON_DOWN	Trigger when the cursor is in the pixel bar and the right mouse button is depressed.
PB_MOUSE_RBUTTON_UP	Trigger when the cursor is in the pixel bar and the right mouse button is released.
PB_MOUSE_MBUTTON_DOWN	Trigger when the cursor is in the pixel bar and the middle mouse button is depressed.
PB_MOUSE_MBUTTON_UP	Trigger when the cursor is in the pixel bar and the middle mouse button is released.
PB_MOUSE_LDBL_CLICK	Trigger when the cursor is in the pixel bar and there is a double click on the left button.

PB_MOUSE_RDBL_CLICK	Trigger when the cursor is in the pixel bar and there is a double click on the right button.
PB_MOUSE_MDBL_CLICK	Trigger when the cursor is in the pixel bar and there is a double click on the middle button.
PB_MOUSE_MOVEMENT	Trigger when the cursor is in the pixel bar and its position changes.
PB_MOUSE_LEAVE	Trigger when the cursor leaves the pixel bar.

fpCallback

The function to call when the specified mouse event occurs. The function returns a status and has one parameter that is a pointer to the mouse event data structure.

Returns

PB_SUCCESS	Function was successful.
PB_ERROR	An unrecoverable error occurred.

lpParam

lpParam is a pointer to the PBMouseEvent data structure. The pixel bar, the mouse event, and if applicable, the x and y locations of the mouse event are reported.

Notes

The issue of mouse interactions is being researched.

Pixelbar_ClearMouseEvent

UINT32 Pixelbar_ClearMouseEvent (UINT32 PixelbarId, UINT32 MouseEventId)

Pixelbar_ClearMouseEvent() clears a previously requested set of mouse events. When the mouse event set is cleared, the Pixel Bar software will no longer service the set.

- Returns one of these values:

PB_SUCCESS	Function was successful.
------------	--------------------------

An unrecoverable error occurred.

Parameters

Pixelbar_Id

The id of the pixel bar.

MouseEventId

The Id of the mouse event set. This value was returned by a previous call to Pixelbar SetMouseEvent.

Notes

The issue of mouse interactions is being researched.

Pixelbar_EnumerateMouseEvents

```

UINT32 Pixelbar_EnumerateMouseEvent (UINT32 PixelbarId, UINT32  

*lpMouseEventIdCount, UINT32 **lpMouseEventIds, UINT32  

*lpMouseEventIdsReturned, UINT32 *lpszData, UINT32  

szOutputBuffer)

```

`Pixelbar_EnumerateMouseEvents()` returns the number of mouse events and the list of mouse events that have been set for the specified pixel bar.

- Returns one of these values:

Function was successful.

An unrecoverable error occurred.

The provided output buffer is too small to contain all the requested information. The information returned in the output buffer is not guaranteed to be coherent or valid. The function should be called again with an output buffer of the appropriate size.

Parameters

PixelbarId

The id of the pixel bar.

lpMouseEventIdCount

The address of the UINT32 where the count of mouse events will be returned.

lpMouseEventIds

The address of an array of UINT32 that hold the Ids of Mouse Event sets.

lpMouseEventIdsReturned

lpMouseEventsReturned is the address of an UINT32 where the count of mouse event ids, which are returned in lpMouseEventIds, will be written. Unless the output buffer pointed to by lpMouseEventIds is too small; the value in lpMouseEventsReturned is the same as in lpMouseEventIdCount.

lpszData

The address of the size of the requested data. If the value is larger than `szOutputBuffer`, then the provided buffer is too small, and the error code `PB_BUFFER_TOO_SMALL` is returned by the function.

szOutputBuffer

The size of the output buffer pointed to by `lpMouseEventIds`, in bytes. This will allow the Pixel Bar software to refrain from overflowing the buffer.

Notes

None.

Pixelbar_QueryMouseEvent

**UINT32 Pixelbar_QueryMouseEvent (UINT32 PixelbarId, UINT32
MouseEventId, PBMouseEventInfo *lpMouseEventInfo, UINT32
*lpszData, UINT32 szOutputBuffer)**

Pixelbar QueryMouseEvent() returns the mouse event state for the specified Mouse Event ID.

- Returns one of these values:

PB_SUCCESS	Function was successful.
PB_ERROR	An unrecoverable error occurred.
PB_BUFFER_TOO_SMALL	The provided output buffer is too small to contain all the requested information. The information returned in the output buffer is not guaranteed to be coherent or valid. The function should be called again with an output buffer of the appropriate size.

Parameters

PixelbarId

The id of the pixel bar.

MouseEventId

The Id of the Mouse Event instance.

lpMouseEventInfo

The address of a `PBMouseEventInfo` structure that the mouse event information is returned in.

lpszData

The address of the size of the requested data. If the value is larger than `szOutputBuffer`, then the provided buffer is too small, and the error code `PB_BUFFER_TOO_SMALL` is returned by the function.

szOutputBuffer

The size of the output buffer pointed to by `lpMouseEventInfo`, in bytes. This will allow the Pixel Bar software to refrain from overflowing the buffer.

Notes

The issue of mouse interactions is being researched.

Pixelbar_EnableMouse

UINT32 Pixelbar_EnableMouse (UINT32 PixelbarId)

Pixelbar EnableMouse enables the display of the mouse cursor icon in the specified pixel bar.

Returns one of these values:

PB_SUCCESS	Function was successful.
PB_ERROR	An unrecoverable error occurred.

Parameters

Pixelbar Id

The id of the pixel bar.

Notes

None.

Pixelbar_DisableMouse

UINT32 Pixelbar DisableMouse (UINT32 PixelbarId)

`Pixelbar_DisableMouse` disables the display of the mouse cursor icon in the specified pixel bar.

Returns one of these values:

<code>PB_SUCCESS</code>	Function was successful.
<code>PB_ERROR</code>	An unrecoverable error occurred.

Parameters

Pixelbar_Id

The id of the pixel bar.

Notes

None.

Pixelbar_QueryDisplayInfo

UINT32 Pixelbar_QueryDisplayInfo (UINT32 DisplayId, PBDisplayInfo *lpDisplayInfo, UINT32 *lpszData, UINT32 szOutputBuffer)

`Pixelbar_QueryDisplayInfo()` queries the pixel bar software for the characteristics and current state of the specified display.

- Returns one of these values:

<code>PB_SUCCESS</code>	Function was successful.
<code>PB_ERROR</code>	An unrecoverable error occurred.
<code>PB_BUFFER_TOO_SMALL</code>	The provided output buffer is too small to contain all the requested information. The information returned in the output buffer is not guaranteed to be coherent or valid. The function should be called again with an output buffer of the appropriate size.

Parameters

DisplayId

The display to query.

lpDisplayInfo

The address of the structure where the display info is returned.

lpSzData

The address of the size of the requested data. If the value is larger than `szOutputBuffer`, then the provided buffer is too small, and the error code `PB_BUFFER_TOO_SMALL` is returned by the function.

szOutputBuffer

The size of the output buffer pointed to by `lpDisplayInfo`, in bytes. This will allow the Pixel Bar software to refrain from overflowing the buffer.

Notes

The physical resolution doesn't include the scanlines used by pixel bars in `MODE_OVERSCAN`.

Pixelbar_DisplayColor

UINT32 Pixelbar_DisplayColor (UINT32 PixelbarId, UINT32 DestX, UINT32 DestY, UINT32 Xext, UINT32 Yext, UINT32 Color)

`Pixelbar_DisplayColor()` specifies a single color value in the current display format that the specified rectangle in the pixel bar is to be colored with.

- Returns one of these values:

`PB_SUCCESS`

Function was successful.

`PB_ERROR`

An unrecoverable error occurred.

Parameters

PixelbarId

The pixel bar to color.

DestX

The x origin, with respect, with respect to the pixel bar origin, of the rectangle in the pixel bar that is to be colored. This value must be within the bounds of the destination pixel bar

DestY

The y origin, with respect, with respect to the pixel bar origin, of the rectangle in the pixel bar that is to be colored. This value must be within the bounds of the destination pixel bar

Xext

The width of the rectangle to be colored. The sum of `DestX` and `Xext` must be less than the width of the destination pixel bar.

Yext

The height of the rectangle to be colored. The sum of DestY and Yext must be less than the height of the destination pixel bar

Color

A single pixel's worth of color value to color the specified rectangle within the pixel bar. The color is in the same format as the screen. For 8 bits per pixel resolutions, only the least significant 8 bits have valid data; for 16 bits per pixel resolutions, only the least significant 16 bits have valid data; and for 24 bits per pixel resolutions, only the least significant 24 bits have valid data. The remaining bits should be 0 (zero). For 32 bits per pixel resolutions, the most significant 8 bits may be set to an Alpha value, which may or may not affect the displayed image, depending on the video device.

For palettized mode displays (e.g. 8 bits per pixel), the pixel bar software currently does not provide a facility to load the color palette, and thus whatever color palette is loaded in the system will be used.

Notes

The pixel bar coordinate system is the same kind as that of the display, with a different origin. For Windows pixel bars, the origin is at the upper left corner of the pixel bar, and the x values increase from left to right, and the y values increase from top to bottom.

Pixelbar_DisplayBitmap

UINT32 Pixelbar_DisplayBitmap (UINT32 PixelbarId, UINT32 SrcX, UINT32 SrcY, UINT32 DestX, UINT32 DestY, UINT32 Xext, UINT32 Yext, UINT32 szBitmap, UINT32 SrcStride, VOID *lpBitmap)

Pixelbar_DisplayBitmap() displays the specified bitmap onto the specified rectangle in the pixel bar.

- Returns one of these values:

PB_SUCCESS

Function was successful.

PB_ERROR

An unrecoverable error occurred.

Parameters

PixelbarId

The destination pixel bar.

SrcX

The x origin, with respect, with respect to the source bitmap origin, of the source rectangle in the source bitmap. The origin is at the upper left corner of the bitmap. This value must be within the bounds of the source bitmap.

SrcY

The y origin, with respect, with respect to the source bitmap origin, of the source rectangle in the source bitmap. The origin is at the upper left corner of the bitmap. This value must be within the bounds of the source bitmap.

 $DestX$

The x origin, with respect, with respect to the pixel bar origin, of the destination rectangle. This value must be within the bounds of the destination pixel bar.

 $DestY$

The x origin, with respect, with respect to the pixel bar origin, of the destination rectangle. This value must be within the bounds of the destination pixel bar.

 X_{ext}

The width of the destination rectangle. The sum of SrcX and Xext must be less than the width of the source bitmap. Also the sum of DestX and Xext must be less than the width of the destination pixel bar.

$$Y_{ext}$$

The height of the destination rectangle. The sum of SrcY and Yext must be less than the height of the source bitmap. Also the sum of DestY and Yext must be less than the height of the destination pixel bar.

szBitmap

The size (in the number of bytes) of the source bitmap.

SrcStride

The number of bytes in each horizontal line of the source bitmap.

lpBitmap

The address of the source bitmap. The bitmap must be in the same screen format (e.g. number of bits per pixel, bits per color, etc.) as the destination display.

Notes

The pixel bar coordinate system is the same kind as that of the display, with a different origin. For Windows pixel bars, the origin is at the upper left corner of the pixel bar, and the x values increase from left to right, and the y values increase from top to bottom.

For palettized mode displays (e.g. 8 bits per pixel), the pixel bar software currently does not provide a facility to load the color palette, and thus whatever color palette is loaded in the system will be used.

Pixelbar_SetDisplayEvent

UINT32 Pixelbar_SetDisplayEvent (UINT32 DisplayId, UINT32 DisplayEvent, UINT32 (*fpCallback)(PBDisplayEvent *lpParam))

Pixelbar_SetDisplayEvent provides the application with a method to be asynchronously notified of various xSides events. The notification is performed via function callbacks.

- Returns one of these values:

PB_SUCCESS	Function was successful.
PB_ERROR	An unrecoverable error occurred.

Parameters

DisplayId

The id of the display device. Currently, only display id 0 (zero) is supported. For the **PB_EVENT_DRIVER_EXIT** event, only display id 0 is valid.

DisplayEvent

The display event that triggers the callback function. It may be one of the following:

PB_EVENT_DRIVER_UNDEFINED	Not currently used, but reserved for future use.
PB_EVENT_DRIVER_ERROR	Trigger when the xSides Driver encounters an error condition. An error code is returned. The error codes include: XSD_FATAL_ERROR A fatal error in the xSides Driver has been encountered. XSD_CONTROL_NOT_FOUND The video driver's Control() function cannot be found. This is a fatal error for the xSides software running on Windows 9x. XSD_DRIVER_HOOK_FAILED The attempt to hook the video driver for Driver Hook Overscan has failed.
PB_EVENT_SCREEN_SAVER_ON	Trigger when screen saver starts running.
PB_EVENT_SCREEN_SAVER_OFF	Trigger when screen saver stops running.

PB_EVENT_DOS_FULL_SCREEN_ON	Trigger when a DOS shell goes to full screen mode.
PB_EVENT_DOS_FULL_SCREEN_OFF	Trigger when a DOS shell ends its full screen mode.
PB_EVENT_DD_FULL_SCREEN_ON	Trigger when a DirectDraw window goes to full screen mode.
PB_EVENT_DD_FULL_SCREEN_OFF	Trigger when a DirectDraw window ends its full screen mode.
PB_EVENT_MONITOR_ON	Trigger when the display monitor is turned on.
PB_EVENT_MONITOR_OFF	Trigger when the display monitor is turned off, e.g. for power saver mode.
PB_EVENT_RESOLUTION_CHANGE	Trigger when the display resolution is changed, and return the new display resolution.
PB_EVENT_DD_FAILURE	Trigger when Direct Draw encounters an unrecoverable error.
PB_EVENT_MSG_MODE_BEGIN	Trigger when the message mode (aka "Blue Screen of Death") occurs.
PB_EVENT_MSG_MODE_END	Trigger when the message mode (aka "Blue Screen of Death") has ended.

fpCallback

The function to call when the specified display event occurs. The function returns a status and has one parameter that is a pointer to the display event data structure.

Returns

PB_SUCCESS	Function was successful.
PB_ERROR	An unrecoverable error occurred.

lpParam

lpParam is a pointer to the PBDisplayEvent data structure. The display id, the display event, and if applicable, new screen resolutions are contained in the structure.

Notes

Pixelbar_ClearDisplayEvent

UINT32 Pixelbar_ClearDisplayEvent (UINT32 DisplayId, UINT32 DisplayEvent)

Pixelbar_ClearDisplayEvent() clears a previously requested display event. When the display event set is cleared, the Pixel Bar software will no longer service the event.

- Returns one of these values:

PB_SUCCESS	Function was successful.
PB_ERROR	An unrecoverable error occurred.

Parameters

DisplayId

The id of the display device. Currently, only display id 0 (zero) is supported. For the **PB_EVENT_DRIVER_EXIT** event, only display id 0 is valid.

DisplayEvent

The display event being cleared, which was previously specified and associated with a callback function by `Pixelbar_SetDisplayEvent()`. It may be one of the following:

PB_EVENT_DRIVER_UNDEFINED	Not currently used, but reserved for future use.
PB_EVENT_DRIVER_ERROR	Trigger when the xSides Driver encounters an error condition. An error code is returned. The error codes include:
XSD_FATAL_ERROR	A fatal error in the xSides Driver has been encountered.
XSD_CONTROL_NOT_FOUND	The video driver's Control() function cannot be found. This is a fatal error for the xSides software running on Windows 9x.
XSD_DRIVER_HOOK_FAILED	The attempt to hook the video driver for Driver Hook Overscan has failed.

PB_EVENT_SCREEN_SAVER_ON	Trigger when screen saver starts running.
PB_EVENT_SCREEN_SAVER_OFF	Trigger when screen saver stops running.
PB_EVENT_DOS_FULL_SCREEN_ON	Trigger when a DOS shell goes to full screen mode.
PB_EVENT_DOS_FULL_SCREEN_OFF	Trigger when a DOS shell ends its full screen mode.
PB_EVENT_DD_FULL_SCREEN_ON	Trigger when a DirectDraw window goes to full screen mode.
PB_EVENT_DD_FULL_SCREEN_OFF	Trigger when a DirectDraw window ends its full screen mode.
PB_EVENT_MONITOR_ON	Trigger when the display monitor is turned on.
PB_EVENT_MONITOR_OFF	Trigger when the display monitor is turned off, e.g. for power saver mode.
PB_EVENT_RESOLUTION_CHANGE	Trigger when the display resolution is changed, and return the new display resolution.
PB_EVENT_DD_FAILURE	Trigger when Direct Draw encounters an unrecoverable error.
PB_EVENT_MSG_MODE_BEGIN	Trigger when the message mode (aka "Blue Screen of Death") occurs.
PB_EVENT_MSG_MODE_END	Trigger when the message mode (aka "Blue Screen of Death") has ended.

Notes

Pixelbar_EnumerateDisplayEvents

UINT32 Pixelbar_EnumerateDisplayEvents (UINT32 DisplayId, UINT32 *lpDisplayEventIdCount, UINT32 **lpDisplayEvents, UINT32 *lpDisplayEventsReturned, UINT32 *lpSzData, UINT32 szOutputBuffer)

Pixelbar_EnumerateDisplayEvents() returns the number of display events and the list of display events that have been set for the specified display device.

- | | |
|---------------------|---|
| PB_SUCCESS | Function was successful. |
| PB_ERROR | An unrecoverable error occurred. |
| PB_BUFFER_TOO_SMALL | The provided output buffer is too small to contain all the requested information. The information returned in the output buffer is not guaranteed to be coherent or valid. The function should be called again with an output buffer of the appropriate size. |

DisplayId

lpDisplayEventCount

lpDisplayEvents

lpDisplayEventsReturned

lpszData

szOutputBuffer

Notes

Pixelbar_QuerySupport

UINT32 Pixelbar_QuerySupport (UINT32 DisplayId, UINT32 xSidesMode, UINT32 XRes, UINT32 YRes, UINT32 BitsPerPixel, UINT32 Edge, PSize *lpPSize, UINT32 *lpszData, UINT32 szOutputBuffer)

Pixelbar_QuerySupport() returns the maximum pixel bar size(s) for the specified xSides Mode, display resolution and display edge. A capability structure is returned.

- Returns one of these values:

PB_SUCCESS	Function was successful.
PB_ERROR	An unrecoverable error occurred.

Parameters

DisplayId

The id of the display device. Currently, only display id 0 (zero) is supported. For the **PB_EVENT_DRIVER_EXIT** event, only display id 0 is valid.

xSidesMode

The xSides mode that the pixel bar is to operate in. May be one of the following:

- **MODE_SHARE** Share mode -- shares the current display resolution with the desktop by decreasing the desktop size.
- **MODE_STEPUP** Step Up mode -- increases the physical resolution of the display, but keep the desktop at the current resolution and use the difference for displaying the pixel bar.
- **MODE_STEPDOWN** Step Down mode -- maintains the physical resolution of the display, but force the desktop to the next smaller defined resolution and use the difference for displaying the pixel bar.
- **MODE_OVERSCAN** Overscan mode -- reprograms the video hardware to provide extra scanlines at the bottom of the screen to display the pixel bar. This mode currently is valid only for pixel bars at the bottom or top edge of the display.

Note: The interaction of pixel bars running in different modes is currently undefined.

XRes

The horizontal resolution of the display.

YRes

The vertical resolution of the display.

BitsPerPixel

The number of bits per pixel.

Edge

Edge specifies which edge of the display surface the pixel bar will be displayed on. May be one or more of the following:

PB_TOP_EDGE	The top edge.
PB_BOTTOM_EDGE	The bottom edge.
PB_RIGHT_EDGE	The right edge. This edge is not valid if the pixel bar is in Overscan mode.
PB_LEFT_EDGE	The left edge. This edge is not valid if the pixel bar is in Overscan mode.

lpPbSize

The address of a PbSize structure that holds the size of the maximum pixel bar size. Values of 0 in either the width or height fields indicate that the xSides Driver cannot support a pixel bar..

lpSzData

The address of the size of the requested data. If the value is larger than szOutputBuffer, then the provided buffer is too small, and the error code PB_BUFFER_TOO_SMALL is returned by the function.

szOutputBuffer

The size of the output buffer pointed to by lpPbSize, in bytes. This will allow the Pixel Bar software to refrain from overflowing the buffer.

Notes**Pixelbar_EnumerateFileInformation**

UINT32 Pixelbar_EnumerateFileInformation (UINT32 *lpTotalFileCount, PBFileInfo *lpFileInfo, UINT32 *lpNumFilesReturned, UINT32 *lpDataSizeReturned, UINT32 *lpSzData, UINT32 szOutputBuffer)

Pixelbar_EnumerateFileInformation() returns the names and versions of the currently installed xSides driver files.

- Returns one of these values:

PB_SUCCESS	Function was successful.
------------	--------------------------

PB_ERROR

An unrecoverable error occurred.

Parameters*lpTotalFileCount*

Pointer to the total number of file entries.

lpFileInfo

Address of the buffer where the file information is to be placed.

lpNumFilesReturned

Pointer to the number of returned file entries.

*lpDataSizeReturned*Address where the amount of data -- in number of bytes -- that is returned (in the buffer specified by *lpFileInfo*) is placed.*lpData*

Address where the total amount of data -- in number of bytes -- available for the file information is placed.

*szOutputBuffer*Size of the buffer pointed to by *lpFileVer*.**Notes****Data Definitions**

The data types used in the Pixel Bar API are defined here. These definitions assume a Little Endian (aka Intel) byte ordering where the most significant byte has the highest address. For the structures depicted below, the most significant byte is on the left, and the least significant byte is on the right, with each line depicting 32 bits (4 bytes).

Integer Types

UINT8	Unsigned 8 bit integer.
UINT16	Unsigned 16 bit integer.
UINT32	Unsigned 32 bit integer.
UINT64	Unsigned 64 bit integer.
INT8	Signed 8 bit integer.

INT16	Signed 16 bit integer.
INT32	Signed 32 bit integer.
INT64	Signed 64 bit integer.

Macros

Return Codes:

PB_SUCCESS	0
PB_ERROR	1
PB_DISPLAY_ID_OUT_OF_BOUNDS	2
PB_DISPLAY_NOT_INITIALIZED	3
PB_INVALID_EDGE_FOR_MODE_OVERSCAN	4
PB_DISPLAY_EDGE_ALREADY_HAS_PIXELBAR	5
PB_PIXELBAR_ID_OUT_OF_BOUNDS	6
PB_PIXELBAR_NOT_CREATED	7
PB_INVALID_EDGE_FOR_MODE_SHARE	8
PB_INVALID_MODE	9
PB_INVALIDATED_ID	10
PB_BAD_PARAMETERS	11
PB_ACCVIO	12
PB_DISPLAY_BUSY	13
PB_UNSUPPORTED_RESOLUTION	14
PB_XSIDES_NOT_INSTALLED	15
PB_SUCCESS_NO_RES_CHANGE	16
PB_BUFFER_TOO_SMALL	17
PB_INSUFFICIENT_RESOURCES	18
PB_ALREADY_DONE	19
PB_PIXELBAR_DEACTIVE	20
PB_DISPLAY_REGION_OUT_OF_BOUNDS	21
PB_VERSION_MISMATCH	22
PB_XSIDES_SUPPORTED	23

xSides Modes:

MODE_OVERSCAN	0	
MODE_SHARE	1	
MODE_STEPUP	2	
MODE_STEPDOWN	3	
MODE_AUTO	4	// Overscan if possible, otherwise Share.

Status Flags:

// Why aren't these called `*_Masks`?

PB_STATUS_ACTIVE	1	// Bit 0
PB_STATUS_EXACT_SIZE	2	// Bit 1
PB_STATUS_INVALID	4	// What is this used for?
PB_STATUS_VALID_RES	8	
PB_STATUS_MOUSE_ENABLED	16	// What is this?
PB_STATUS_DISPLAYED	9	// What is this?

Display Edges:

```
// Change from
```

bitmasks to ordinals.

PB_BOTTOM_EDGE	0
PB_RIGHT_EDGE	1
PB_TOP_EDGE	2
PB_LEFT_EDGE	3

Display Edge Masks:

PB_BOTTOM_EDGE_MASK	1
PB_RIGHT_EDGE_MASK	2
PB_TOP_EDGE_MASK	4
PB_LEFT_EDGE_MASK	8

Mouse Events:

PB_MOUSE_LBUTTONDOWN	1
PB_MOUSE_LBUTTON_UP	2
PB_MOUSE_RBUTTONDOWN	4
PB_MOUSE_RBUTTON_UP	8

PB_MOUSE_MBUTTON_DOWN	16
PB_MOUSE_MBUTTON_UP	32
PB_MOUSE_LDBL_CLICK	64
PB_MOUSE_RDBL_CLICK	128
PB_MOUSE_MDBL_CLICK	256
PB_MOUSE_MOVEMENT	512
PB_MOUSE_LEAVE	1024

Display Events:

PB_EVENT_DRIVER_UNDEFINED	0
PB_EVENT_DRIVER_ERROR	1
PB_EVENT_SCREEN_SAVER_ON	2
PB_EVENT_SCREEN_SAVER_OFF	3
PB_EVENT_DOS_FULL_SCREEN_ON	4
PB_EVENT_DOS_FULL_SCREEN_OFF	5
PB_EVENT_DD_FULL_SCREEN_ON	6
PB_EVENT_DD_FULL_SCREEN_OFF	7
PB_EVENT_MONITOR_ON	8
PB_EVENT_MONITOR_OFF	9
PB_EVENT_RESOLUTION_CHANGE	10
PB_EVENT_DD_FAILURE	11
PB_EVENT_MSG_MODE_BEGIN	12
PB_EVENT_MSG_MODE_END	13

xSides Driver Error

XSD_FATAL_ERROR	0
XSD_CONTROL_NOT_FOUND	1
XSD_DRIVER_HOOK_FAILED	2

XSD_FILE_TYPE:

PB_DLL	10
PB_XSIDES_DLL	11
PB_XSIDESK_SYS	12
PB_XSD9XDLL_DLL	13
PB_XSIDES_VXD	14

Structure Types

PBConfiguration

UINT16 Version	UINT16 Size
UINT32 DisplayId	
UINT32 PixelBarMask	
UINT32 MaxOverscanWidth	
UINT32 MaxOverscanHeight	
UINT32 MaxOverscanBPP	
UINT32 EdgeFlags	

Version

The structure version. This occupies the third and fourth bytes of the structure (bytes 2 and 3).

Size

The structure size, in bytes, including the version and size fields. This occupies the first two bytes of the structure (bytes 0 and 1).

DisplayId

The display to apply the configuration information.

PixelBarMask

A bitmask indicating the pixel bars that will be displayed. A union of the pixel bar edges that will be used during the current xSides session.

MaxOverscanWidth

The maximum cumulative width, in pixels, of the overscan regions. A value of 0 indicates that overscan will not be used.

MaxOverscanHeight

The maximum cumulative height, in pixels, of the overscan regions. A value of 0 indicates that overscan will not be used.

MaxOverscanBPP

The maximum pixel resolution that overscan pixel bars will be displayed in. Can be 8, 16, 24 or 32. A value of 0 indicates that overscan mode will not be used.

EdgeFlags

A bitfield indicating which of the 4 sides will have pixel bars. A union of 1 or more of the following:

PB_BOTTOM_EDGE

PB_RIGHT_EDGE

PB_TOP_EDGE

PB_LEFT_EDGE

PBDisplayInfo

UINT16 Version	UINT16 Size
UINT32 DisplayId	
UINT32 HwRezX	
UINT32 HwRezY	
UINT32 TotalRezX	
UINT32 TotalRezY	
UINT32 DesktopRezX	
UINT32 DesktopRezY	
UINT32 BitsPerPixel	

UINT32 RefreshFrequency
UINT32 InterlaceFlag
UINT32 Stride

Version

The structure version. This occupies the third and fourth bytes of the structure (bytes 2 and 3).

Size

The structure size, in bytes, including the version and size fields. This occupies the first two bytes of the structure (bytes 0 and 1).

DisplayId

The display that the information is being returned for.

HwRezX

The number of pixels along the X axis that are being displayed by the video driver. This number doesn't include any overscan regions.

HwRezY

The number of pixels along the Y axis that are being displayed by the video driver. This number doesn't include any overscan regions.

TotalRezX

The number of pixels along the X axis that are being displayed by the video hardware, including all overscans regions.

TotalRezY

The number of pixels along the Y axis that are being displayed by the video hardware, including all overscan regions.

DesktopRezX

The number of pixels along the X axis that are being used by the desktop. This excludes all regions used by active pixel bars.

DesktopRezY

The number of pixels along the Y axis that are being used by the desktop. This excludes all regions used by active pixel bars.

BitsPerPixel

The current pixel resolution of the display, expressed as the number of bits per pixel. It is assumed that the display is a packed flat, non-banked frame buffer.

RefreshFrequency

The current refresh frequency of the display.

InterlaceFlag

TRUE (1) if the display is interlaced, otherwise FALSE (0).

Stride

The number of bytes between adjacent scan lines in the video frame buffer, including the bytes in the displayed scan line. If not known, then it should be filled with 0 (zero).

PBPixelbarState

UINT16 Version	UINT16 Size
UINT32 PixelbarId	
UINT32 DisplayId	
UINT32 XSidesMode	
UINT32 Status	
UINT32 Edge	
UINT32 OriginX	
UINT32 OriginY	
UINT32 Width	
UINT32 Height	

Version

The structure version. This occupies the third and fourth bytes of the structure (bytes 2 and 3).

Size

The structure size, in bytes, including the version and size fields. This occupies the first two bytes of the structure (bytes 0 and 1).

PixelbarId

The Id of the pixel bar.

DisplayId

The Id of the display that the pixel bar is on.

xSidesMode

The mode that the pixel bar is in. May be one of the following:

MODE_OVERSCAN

MODE_SHARE

MODE_STEPUP

MODE_STEPDOWN

MODE_AUTO

Status

The current status of the pixel bar. The following bits are defined and may be set to TRUE (1) or FALSE (0).

PB_STATUS_ACTIVE

PB_STATUS_EXACT_SIZE

Edge

The edge of the display that the pixel bar is located at. May be one of the following:

OriginX

The X origin of the pixel bar, in hardware screen coordinates.

OriginY

The Y origin of the pixel bar, in hardware screen coordinates.

Width

The current width of the pixel bar, in pixels.

Height

The current height of the pixel bar, in pixels.

CURSOR_POS

UINT32 X_Position
UINT32 Y_Position

X_Position

The X_Position of the cursor.

Y_Position

The Y_Position of the cursor.

PBMouseEvent

UINT16 Version	UINT16 Size
UINT32 PixelbarId	
UINT32 MouseEvent	
UINT32 ButtonState	
CURSOR_POS CursorPos	
UINT32 RepeatCount	

Version

The structure version. This occupies the third and fourth bytes of the structure (bytes 2 and 3).

Size

The structure size, in bytes, including the version and size fields. This occupies the first two bytes of the structure (bytes 0 and 1).

PixelbarId

Indicates which pixel bar the mouse event occurred in.

MouseEvent

Indicates the mouse event being reported. May be one of:

- PB_MOUSE_LBUTTON_DOWN
- PB_MOUSE_LBUTTON_UP

- PB_MOUSE_RBUTTON_DOWN
- PB_MOUSE_RBUTTON_UP
- PB_MOUSE_MBUTTONDOWN_DOWN
- PB_MOUSE_MBUTTONDOWN_UP
- PB_MOUSE_LDBL_CLICK
- PB_MOUSE_RDBL_CLICK
- PB_MOUSE_MDBL_CLICK
- PB_MOUSE_MOVEMENT
- PB_MOUSE_LEAVE

ButtonState

What is this?

CursorPos

The position of the cursor relative to the pixel bar origin of the mouse movement. Valid only if the mouse event is PB_MOUSE_MOVEMENT.

RepeatCount

What is this?

PBMouseEventInfo

UINT16 Version	UINT16 Size
UINT32 PixelbarId	
UINT32 MouseEventId	
UINT32 MouseEvent	
UINT32 fpCallbackFunction	

Version

The structure version. This occupies the third and fourth bytes of the structure (bytes 2 and 3).

Size

The structure size, in bytes, including the version and size fields. This occupies the first two bytes of the structure (bytes 0 and 1).

PixelbarId

The Pixel Bar that the Mouse Event set is associated with.

MouseEventId

The Id of the Mouse Event set.

MouseEventId

Indicates the mouse event being reported. May be one or more of:

- PB_MOUSE_LBUTTON_DOWN
- PB_MOUSE_LBUTTON_UP
- PB_MOUSE_RBUTTON_DOWN
- PB_MOUSE_RBUTTON_UP
- PB_MOUSE_MBUTTON_DOWN
- PB_MOUSE_MBUTTON_UP
- PB_MOUSE_LDBL_CLICK
- PB_MOUSE_RDBL_CLICK
- PB_MOUSE_MDBL_CLICK
- PB_MOUSE_MOVEMENT
- PB_MOUSE_LEAVE

fpCallbackFunction

The address of the callback function associated with this Mouse Event set.

PBDisplayEvent

UINT16 Version	UINT16 Size
UINT32 DisplayId	
UINT32 DisplayEvent	
UINT32 SubCode	
PBDisplayInfo NewDisplayInfo UINT32 ErrorCode	

Version

The structure version. This occupies the third and fourth bytes of the structure (bytes 2 and 3).

Size

The structure size, in bytes, including the version and size fields. This occupies the first two bytes of the structure (bytes 0 and 1).

DisplayId

Indicates which display device the display event occurred in.

DisplayEvent

Indicates the display event being reported. May be one of the following:

- PB_EVENT_DRIVER_ERROR
- PB_EVENT_SCREEN_SAVER_ON
- PB_EVENT_SCREEN_SAVER_OFF
- PB_EVENT_DOS_FULL_SCREEN_ON
- PB_EVENT_DOS_FULL_SCREEN_OFF
- PB_EVENT_DD_FULL_SCREEN_ON
- PB_EVENT_DD_FULL_SCREEN_OFF
- PB_EVENT_MONITOR_ON
- PB_EVENT_MONITOR_OFF
- PB_EVENT_RESOLUTION_CHANGE
- PB_EVENT_DD_FAILURE
- PB_EVENT_MSG_MODE_BEGIN

- PB_EVENT_MSG_MODE_END

SubCode

Indicates a sub event code for the Display Event. The default value is 0.

For PB_EVENT_DRIVER_ERROR, the valid subcodes are:

- XSD_FATAL_ERROR
- XSD_CONTROL_NOT_FOUND
- XSD_DRIVER_HOOK_FAILED

NewDisplayInfo

Data reflecting the new state of the display when the event is PB_EVENT_RESOLUTION_CHANGE.

PBPixelbarSize

UINT16 Version	UINT16 Size
UINT32 BaseWidth	
UINT32 BaseHeight	
INT32 OriginX	
INT32 OriginY	
UINT32 Width	
UINT32 Height	

Version

The structure version. This occupies the third and fourth bytes of the structure (bytes 2 and 3).

Size

The structure size, in bytes, including the version and size fields. This occupies the first two bytes of the structure (bytes 0 and 1).

BaseWidth

Copyright © 2000 xSides Corporation

BaseWidth specifies the horizontal number of pixels of the resolution that the pixel bar size applies to. For example, if the size of the pixel bar for the resolution of 640x480 is being specified, then the BaseWidth would be set to 640.

BaseHeight

BaseHeight specifies the vertical number of pixels of the resolution that the pixel bar size applies to. For example, if the size of the pixel bar for the resolution of 640x480 is being specified, then the BaseWidth would be set to 480.

OriginX

OriginX specifies the x position on the display surface, in units of pixels with respect to the Desktop Origin, where the origin of the pixel bar (which is the upper left corner of the pixel bar) is located. The pixel bar origin must be consistent with its screen edge parameter; otherwise an error will be returned. This is a signed value and may be a negative value (which it will be for non-zero width pixel bars on the left edge of the screen.)

OriginY

OriginY specifies the y position on the display surface, in units of pixels with respect to the Desktop Origin, where the origin of the pixel bar (which is the upper left corner of the pixel bar) is located. The pixel bar origin must be consistent with its screen edge parameter; otherwise an error will be returned. This is a signed value and may be a negative value (which it will be for non-zero height pixel bars on the top edge of the screen.)

Width

Width specifies the desired width of the pixel bar, in number of pixels. It may not be larger than the total physical horizontal resolution of the display, which include the desktop and the pixel bars.

Height

Height specifies the desired height of the pixel bar, in number of pixels. It may not be larger than the total physical vertical resolution of the display, which include the desktop and the pixel bars.

PBSize

UINT16 Version	UINT16 Size
UINT32 Width	

UINT32 Height

Version

The structure version. This occupies the third and fourth bytes of the structure (bytes 2 and 3).

Size

The structure size, in bytes, including the version and size fields. This occupies the first two bytes of the structure (bytes 0 and 1).

Width

Width specifies the desired width of the pixel bar, in number of pixels. It may not be larger than the physical horizontal resolution of the display.

Height

Height specifies the desired height of the pixel bar, in number of pixels. It may not be larger than the physical vertical resolution of the display.

PBFileInfo

UINT16 Version	UINT16 Size
XSD_FILE_TYPE edFileType	
UINT32 szFileSpec	
UINT32 szFileVersion	
UINT32 dwFileSpecIndex	
UINT32 dwFileVersionIndex	
char FileData[1]	

Version

The structure version. This occupies the third and fourth bytes of the structure (bytes 2 and 3).

Size

The structure size, in bytes, including the version and size fields. This occupies the first two bytes of the structure (bytes 0 and 1). Since this is a variable size structure, the size field allows the a buffer of PBFileInfo structures to be parsed.

edFileType

Width specifies the desired width of the pixel bar, in number of pixels. It may not be larger than the physical horizontal resolution of the display.

szFileSpec

The number of bytes in the FileSpec, which is a null terminated character string, including the directory path and file name of the xSides driver file. The null terminator is not counted.

szFileVersion

The number of bytes in the FileVersion, which is a null terminated character string, including version information for the xSides driver file. The null terminator is not counted.

dwFileSpecIndex

Index into the.

dwFileVersionIndex

Height specifies the desired height of the pixel bar, in number of pixels. It may not be larger than the physical vertical resolution of the display.

FileData

A character array where the FileSpec and FileVersion information are stored as null terminated strings.

Index

CURSOR_POS	5, 46	PB_BAD_PARAMETERS	37
Data Definitions	36	PB_BOTTOM_EDGE	8, 11, 35, 38, 42
Display Edge Masks	38	PB_BUFFER_TOO_SMALL	15, 16, 17, 20, 21, 22, 24, 32, 33, 35, 37
Display Edges	38	PB_DISPLAY_BUSY	37
Display Events	39	PB_DISPLAY_EDGE_ALREADY_HAS_PIXELBAR	37
INT16	36	PB_DISPLAY_ID_OUT_OF_BOUNDS	37
INT32	36	PB_DISPLAY_NOT_INITIALIZED	37
INT64	37	PB_DISPLAY_REGION_OUT_OF_BOUNDS	38
INT8	36	PB_DLL	41
Integer Types	36	PB_ERROR	37
Macros	37	PB_EVENT_DD_FAILURE	32, 40
MODE_AUTO	7, 10, 38, 45	PB_EVENT_DD_FULL_SCREEN_OFF	29, 31, 40, 50
MODE_OVERSCAN	7, 9, 14, 24, 34, 38, 45	PB_EVENT_DD_FULL_SCREEN_ON	29, 31, 40, 50
MODE_SHARE	7, 9, 14, 34, 38, 45	PB_EVENT_DOS_FULL_SCREEN_OFF	28, 31, 39, 50
MODE_STEPDOWN	7, 9, 15, 34, 38, 45		
MODE_STEPUP	7, 9, 15, 34, 38, 45		
Mouse Events	39		
PB_ACCVIO	37		
PB_ALREADY_DONE	37		

PB_EVENT_DOS_FULL_SCREEN_ON	28, 31, 39, 50	PBConfiguration	5, 41
PB_EVENT_DRIVER_ERROR	28, 30, 39, 50, 51	PBDisplayEvent	5, 27, 30, 49, 51, 53, 54
PB_EVENT_DRIVER_EXIT	28, 30, 33, 34	PBDisplayInfo	2, 24, 42, 49
PB_EVENT_MONITOR_OFF	29, 31, 40, 50	PBDisplayState	5
PB_EVENT_MONITOR_ON	29, 31, 40, 50	PBMouseEvent	5, 17, 19, 46
PB_EVENT_MSG_MODE_BEGIN	29, 40, 50	PBMouseEventInfo	5, 21, 22, 48
PB_EVENT_MSG_MODE_END	29, 40, 50	PBPixelbarState	2, 16, 17, 44
PB_EVENT_RESOLUTION_CHANGE	29, 31, 32, 40, 50, 51	PBPixelBarState	5
PB_EVENT_SCREEN_SAVER_OFF	28, 31, 39, 50	Pixelbar_Activate	3, 14
PB_EVENT_SCREEN_SAVER_ON	28, 31, 39, 50	Pixelbar_ClearDisplayEvent	4, 30
PB_INSUFFICIENT_RESOURCES	37	Pixelbar_ClearMouseEvent	4, 19, 20
PB_INVALID_EDGE_FOR_MODE_OVERSCAN	37	Pixelbar_Create	3
PB_INVALID_EDGE_FOR_MODE_SHARE	37	Pixelbar_CreatePixelBar	9, 10
PB_INVALID_MODE	37	Pixelbar_Deactivate	3, 14
PB_INVALIDATED_ID	37	Pixelbar_Delete	3, 13
PB_LEFT_EDGE	8, 11, 35, 38, 39, 42	Pixelbar_DisplayBitmap	4, 26
PB_MOUSE_LBUTTON_DOWN	18, 39, 47, 48	Pixelbar_DisplayColor	4, 24, 25
PB_MOUSE_LBUTTON_UP	18, 39, 47, 48	Pixelbar_Enumerate	3, 15
PB_MOUSE_LDBL_CLICK	19, 39, 47, 49	Pixelbar_EnumerateDisplayEvents	4, 32, 33, 35
PB_MOUSE_LEAVE	39	Pixelbar_EnumerateMouseEvents	4, 20, 21
PB_MOUSE_MBUTTON_DOWN	18, 39, 47, 48	Pixelbar_Init	3, 5
PB_MOUSE_MBUTTON_UP	18, 39, 47, 49	Pixelbar_Modify	3, 11, 12
PB_MOUSE_MDBL_CLICK	19, 39, 47, 49	Pixelbar_Query	3, 10, 12, 15, 16
PB_MOUSE_MOVEMENT	18, 19, 39, 47, 48, 49	Pixelbar_QueryDisplayInfo	4, 23, 24
PB_MOUSE_RBUTTON_DOWN	18, 39, 47, 48	Pixelbar_QueryMouseEvent	4, 21
PB_MOUSE_RBUTTON_UP	18, 39, 47, 48	Pixelbar_SetDisplayEvent	4, 27, 30
PB_MOUSE_RDBL_CLICK	19, 39, 47, 49	Pixelbar_SetMouseEvent	4, 17, 20, 22, 23
PB_PIXELBAR_DEACTIVE	37	Pixelbar_Uninit	3, 6
PB_PIXELBAR_ID_OUT_OF_BOUNDS	37	Return Codes	37
PB_PIXELBAR_NOT_CREATED	37	Status Flags	38
PB_RIGHT_EDGE	8, 11, 35, 38, 42	Structure Types	41
PB_STATUS_ACTIVE	32, 38	UINT16	36
PB_STATUS_DISPLAYED	38	UINT32	36
PB_STATUS_EXACT_SIZE	38	UINT64	36
PB_STATUS_INACTIVE	32	UINT8	36
PB_STATUS_INVALID	38	XSD_CONTROL_NOT_FOUND	40
PB_STATUS_MOUSE_ENABLED	38	XSD_DRIVER_HOOK_FAILED	40
PB_STATUS_VALID_RES	38	XSD_FATAL_ERROR	40
PB_SUCCESS	37	XSD_FILE_TYPE	41
PB_SUCCESS_NO_RES_CHANGE	37	xSides Driver Error	40
PB_TOP_EDGE	8, 11, 35, 38, 39, 42	xSides Modes	38
PB_UNSUPPORTED_RESOLUTION	37	40	
PB_VERSION_MISMATCH	38	40	
PB_XSD9XDLL_DLL	41	40	
PB_XSIDES_DLL	41	40	
PB_XSIDES_NOT_INSTALLED	37	40	
PB_XSIDES_SUPPORTED	38	40	
PB_XSIDES_VXD	41	40	
PB_XSIDESK_SYS	41	40	

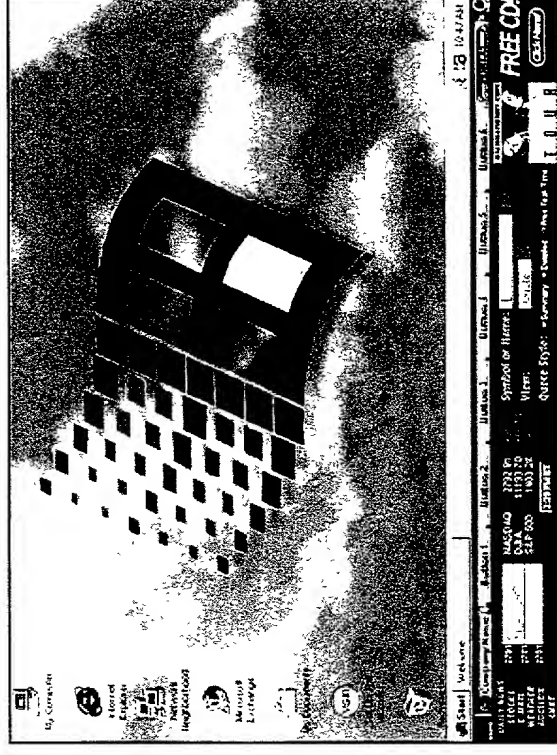
Pixel Bar Header File

[illegible]

xSides Overview

xSides provides a customizable, multi-task application that resides outside the confines of the Windows™ desktop.

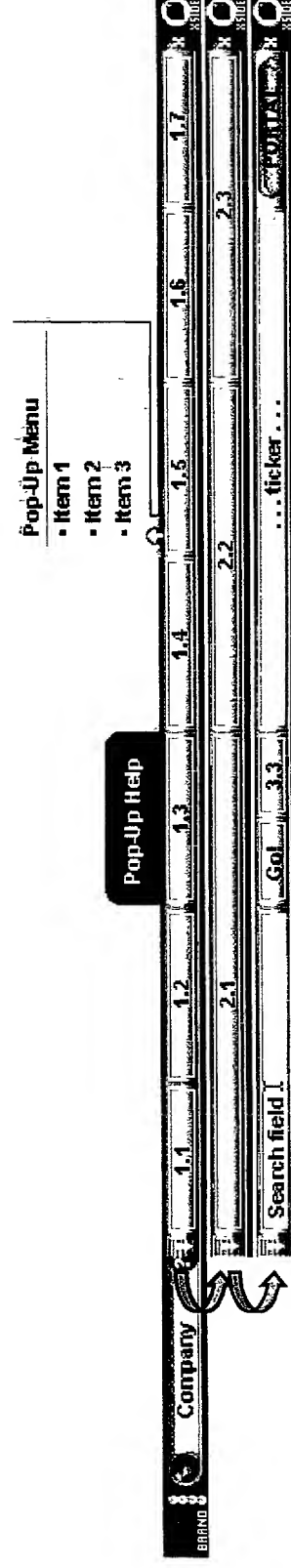
- Buttons, logos, and menus can be customized to create unique sides, using the Level 1 and Level 2 Application Programming Interface (API)
- A Custom side allows for limited customization of special buttons by the user



xSides Side Applications

xSides rotates to allow many sides on one application.

- A side is one row of objects, such as buttons and tickers.
- Sides can be accessed through the rotator buttons or through a navigation menu.
- Objects of one side may be designed to work together as a Web browser, while the objects of another side may be a row of buttons linked to various pages of a company's Web site.



xSides Customization

Buttons, logos, and menus can be customized by the developer to create unique sides using the Level 1 and Level 2 Application Programming Interface (API).

Level 1 API:

- Create new sides by assigning names and paths to standard buttons in the DAT files.
- Standard buttons include clickable buttons, tickers, spacers, and rotators of various widths denoted by their BMP ID#.
- Assign names and paths to cascading menus in a DAT file.
- The content of a DAT file is in XML.

xSides Customization

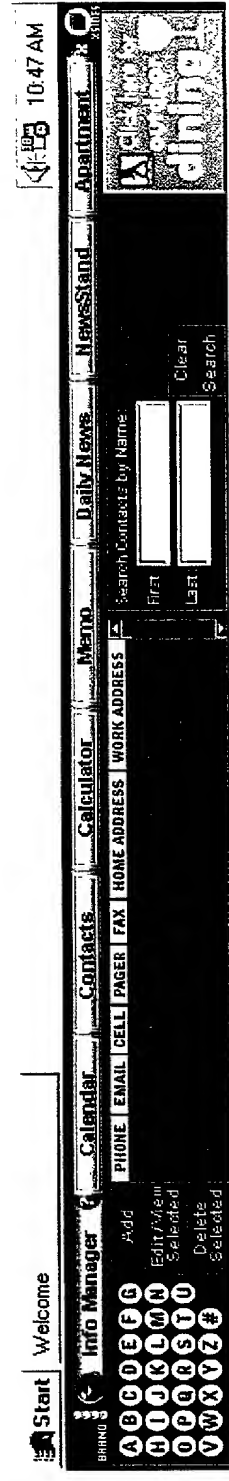
Level 2 API:

- 1. Add functionality to objects, like buttons and tickers, through a set of callable DLL functions and messages.**
- 2. DLL support includes side object resources like bitmaps, palette, and dialogs.**
- 3. Requires a standard Windows development environment**

Portal Overview

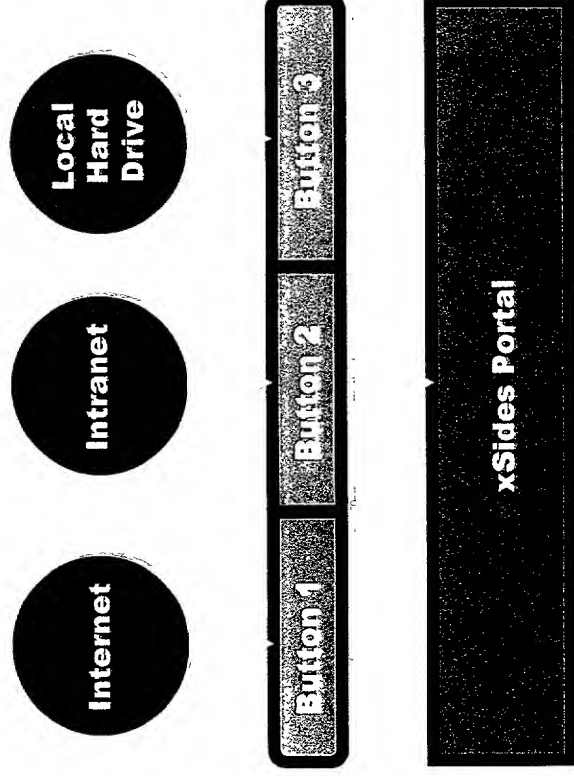
The Portal area functions exclusively with xSides. The xSides Portal launches underneath a side and responds to the xSides commands. Like xSides, the xSides Portal cannot be covered by Windows.

- xSides Portal source can come from the Internet, intranet or locally from the hard disk.
- Portal creation is as simple as creating a Web page.
- Portal height is adjustable.



Portals' Source

The xSides Portal source can come from the Internet, intranet, or locally, from the hard disk. Multiple portals can be assigned to a single product, provided each portal has its own link through a button or menu option. Only one portal can be displayed at a time.



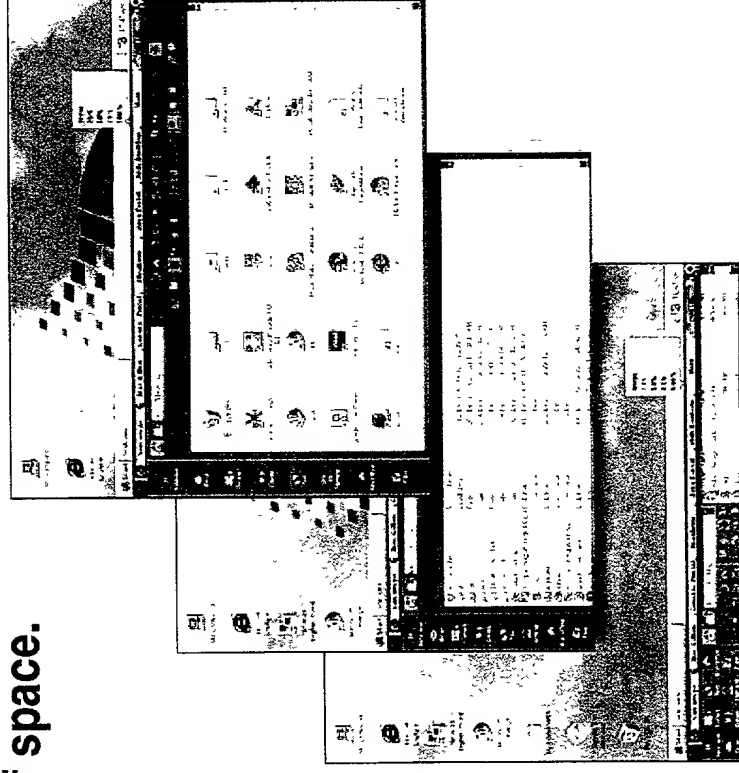
Portal Development Standards

- Server-side data manipulation can be done with any languages or programs as long as you follow Internet Explorer 4.0 standards for HTML, Dynamic HTML, JavaScript and Cascading Style Sheets for client-side output.
- If features are developed using Internet Explorer 5.0 standards, a backward-compatible solution should be in place.
- Cursors will not appear in text and text area elements.
- Some form elements cannot function in a xSides Portal area, but a JavaScript equivalent is available on request.
- Applications like Java applets or Macromedia's Flash are not currently supported.

Portal Adjusting Height

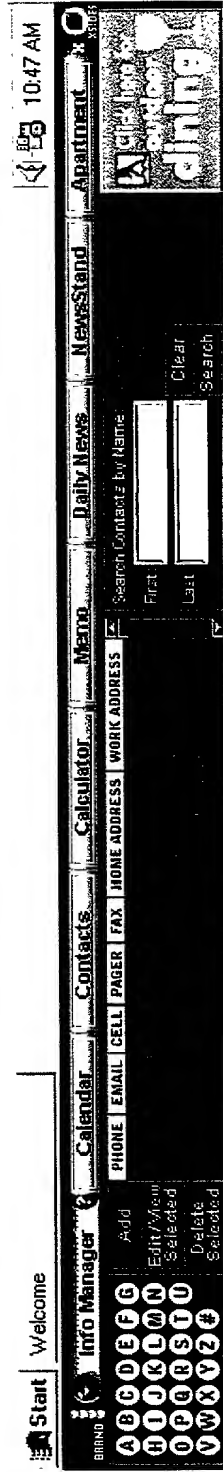
The xSides Portal height can be adjusted for maximum control and efficiency of space.

- Adjust the height in pixels by changing a single value in an INI file.
- Normal xSides Portal height is 70 pixels.
- Portal height can range from 0 (no portal) to maximum vertical screen resolution minus the height of xSides.



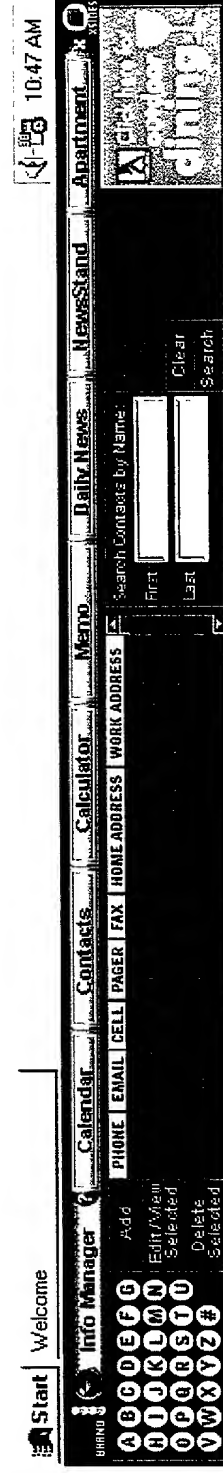
Portal Web-Based Applications

In addition to hosting Web pages, the portal can support applications that run entirely over the Internet, are always up to date, and conserve local hard disk space. The xSides Portal is the perfect environment for running Web-Based Applications.



Portal Web-Based Applications Example: Personal Information Manager (PIM)

The PIM is a time-saving desktop organizer opened from xSides, that functions inside the portal. With the xSides User Registration feature, a user's personal PIM data can be accessed from any computer.

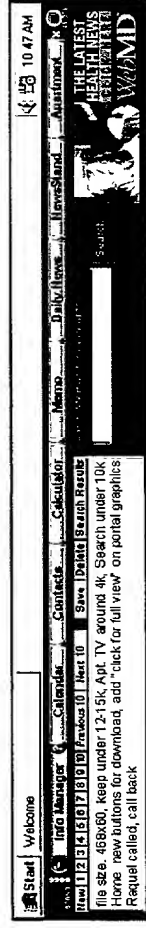


Portal Web-Based Applications

Example: Personal Information Manager (PIM)

PIM features include:

- Address Book
- Calendar
- Calculator
- Memo Feature
- To-Do List
- Daily News
- Newsstand
- Import/Export Feature



User Registration Overview

User Registration is a client/server application that gathers user information for data processing and storage.

- Allows multiple users
- Access your bar, address book, and calendar from multiple computers
- Access to additional features, promotions, and updates
- Easy to use
- Is a basis for Universal Registration

User Registration How Information is Gathered

- HTML forms gather and submit user information.
- Information is submitted to a server for authentication and storage.

User Registration Functionality

- Registration
- Login
- Logout
- Change Profile
- Forgot My Password

AllSides Overview

- Allows sides to be added and removed from a user's xSides product.
- Automatically updates sides as newer versions become available.
- Allows sides to include dependent files.
- Enables a registered user to travel with their xSides configuration.
- Identifies sides installed by a user, the download source for these sides, and the use of those sides by a user.

AlSides Default Configuration

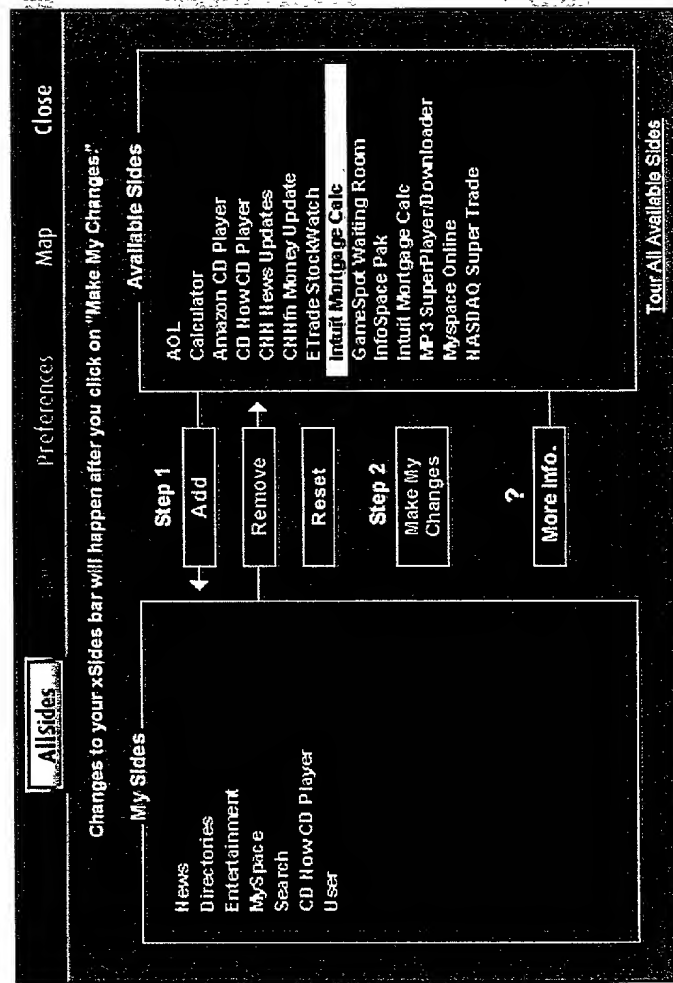
When a user downloads the xSides product for the first time, it includes a default configuration specific to the download source.

The configuration includes:

- Logo (appearing to the left of the bar)
- Base set of sides
- Class assignment
- GUID generation and server registration

AllSides Adding and Removing Sides (1)

The user can add and remove sides by selecting the AllSides dialog from the help menu.



AllSides Adding and Removing Sides (2)

- Available sides displayed in the right list box (based on user's class).
- Installed sides appear on the left.
- When the "Make My Changes" button is pressed, the xSides application will:
 - Update the local copy of the user's configuration file
 - Retrieve the side definition file(s) from the server
 - Post the updated user configuration file to the server
 - Regenerate xSides to display the selected changes

AllSides Future Enhancements

As a future enhancement, sides will be available for download directly from an HTML page.

- Every side that is added to a user's configuration includes a Distribution Vendor ID (DVID).
- DVID allows for flexibility in tracking users' activity on a side, based on the download source.

AllSides Side Installation

- Sides that are downloaded from the server are placed in a temporary folder on the local file system.
- A “package” is defined as a series of files required for a side to be successfully installed (dependent INI or DLL files, for instance).
- Sides are not available for installation until all elements of the package are available locally, and the application version meets the minimum version required by the side.
- Sides are installed by being “promoted” from the temporary folder into the working folder. Side promotion happens automatically when all criteria is met.

AllSides Traveling With xSides

If a user has registered, they can to download their configuration to any workstation running the xSides application by simply logging in using their username and password.

- User configuration file is retrieved from the server
- Any sides that do not already exist on that workstation are installed
- Feature provides a familiar, consistent interface with the application for those users that use multiple workstations frequently (for instance, both at work and at home)
- Changes made on either workstation are automatically synchronized with the server, even if both instances of the application are running simultaneously

AllSides Automatic Update

- Every side is described by a separate file, which includes an embedded date/time stamp.
- The xSides application periodically checks all the sides cached locally against the revisions available on the server.
- New side files are downloaded in the background and automatically installed.
- Sides that have been removed from the user configuration files of all users on the client will be deleted from the local file system.

User configuration files that have been unused for 30 days will also be removed from the local file system to conserve disk space, though the data is still retrievable from the server if needed after 30 days.

AllSides File System Details

- Side files are stored in a separate folder from the user configuration files.
- Side files contain the data required to generate a side, a revision date/time stamp, dependent files (if any) and the minimum executable version (if required).
- DVIDs, sides, user buttons and the order in which all sides appear on xSides are all stored in the user configuration file, along with a revision date/time stamp (GMT).
- All files are stored encrypted.

Communications Layer (CommLayer) Overview

The CommLayer provides xSides dependencies conduit for communication with the xSides server. Each xSides user is identified by Global Unique Identifiers (GUID).

- Client/server architecture.
- XML driven.
- Efficient - all communication is coalesced to minimize Internet traffic.

CommLayer Features (1)

- Ping (scheduled server communication).
- Allows dependent components to register/unregister with CommLayer for ping callbacks.
- Schedules task for dependent components.
- Hot swapping of newly downloaded dependent components (excluding the CommLayer and the xSides executable).

CommLayer Features (2)

- Loading and unloading of dependent components (e.g., IA, Stats, mktplace) at the start and end of every xSides session.
- Auto updating of system files (happens in the background).
- Provides a conduit for communication between dependent components and xSides.
- Extensible.

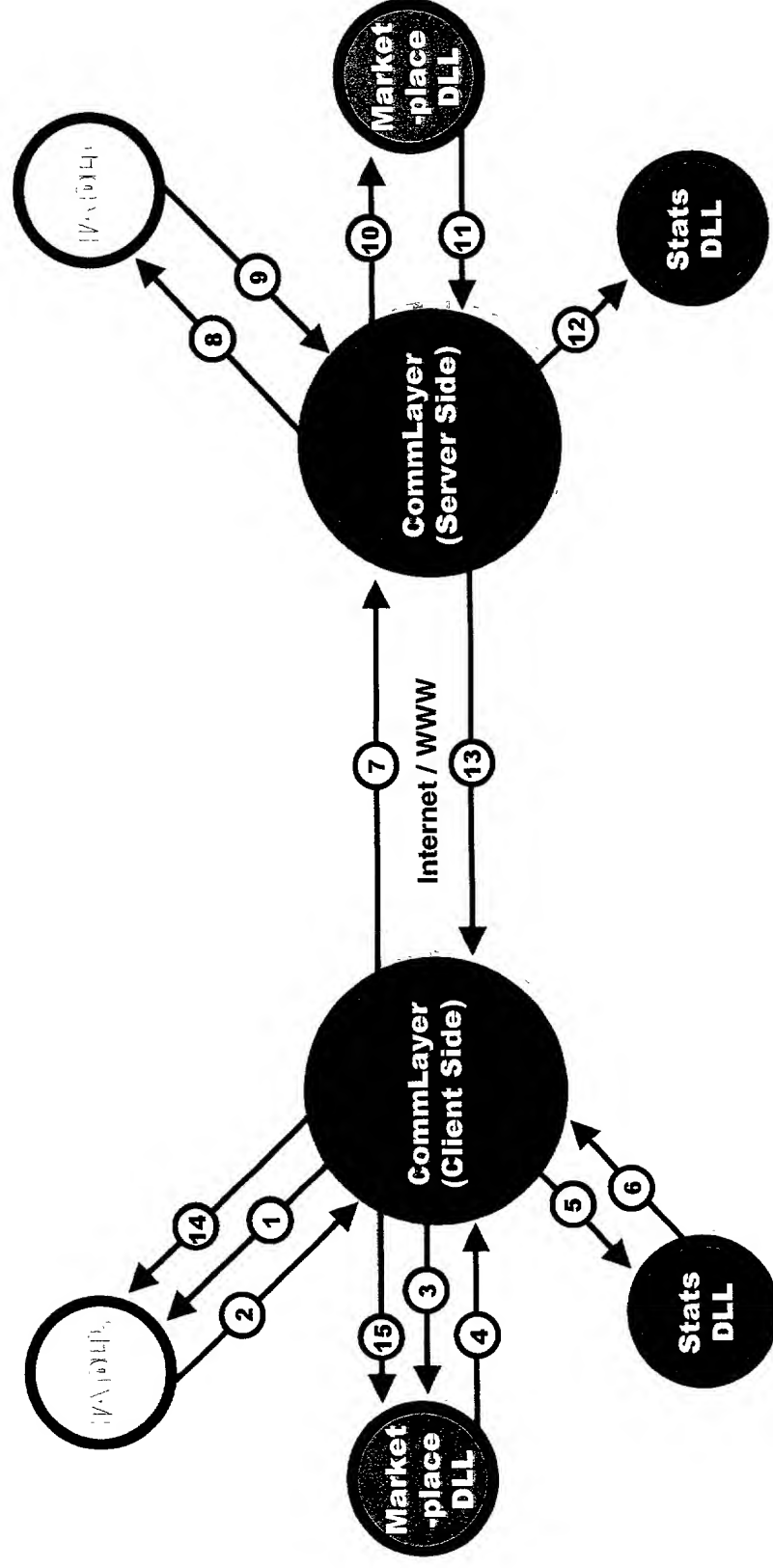
CommLayer Ping Functionality (1)

- Runs on a server-assigned time interval.
- Queries all of the registered DLLs and builds the needed markup which is then prepared to be sent to the xSides server.
- Encrypts the markup.
- Sends the markup to the server (GUID is sent with the ping to identify the user).

CommLayer Ping Functionality (2)

- Server-side CommLayer DLL parses and passes the markup to registered DLLs on server side.
- These registered DLLs process the markup and return their responses back to the server side CommLayer.
- Server encrypts the responses and sends them back to the client-side CommLayer.
- Client decrypts, parses, and passes responses back to appropriate registered DLLs.

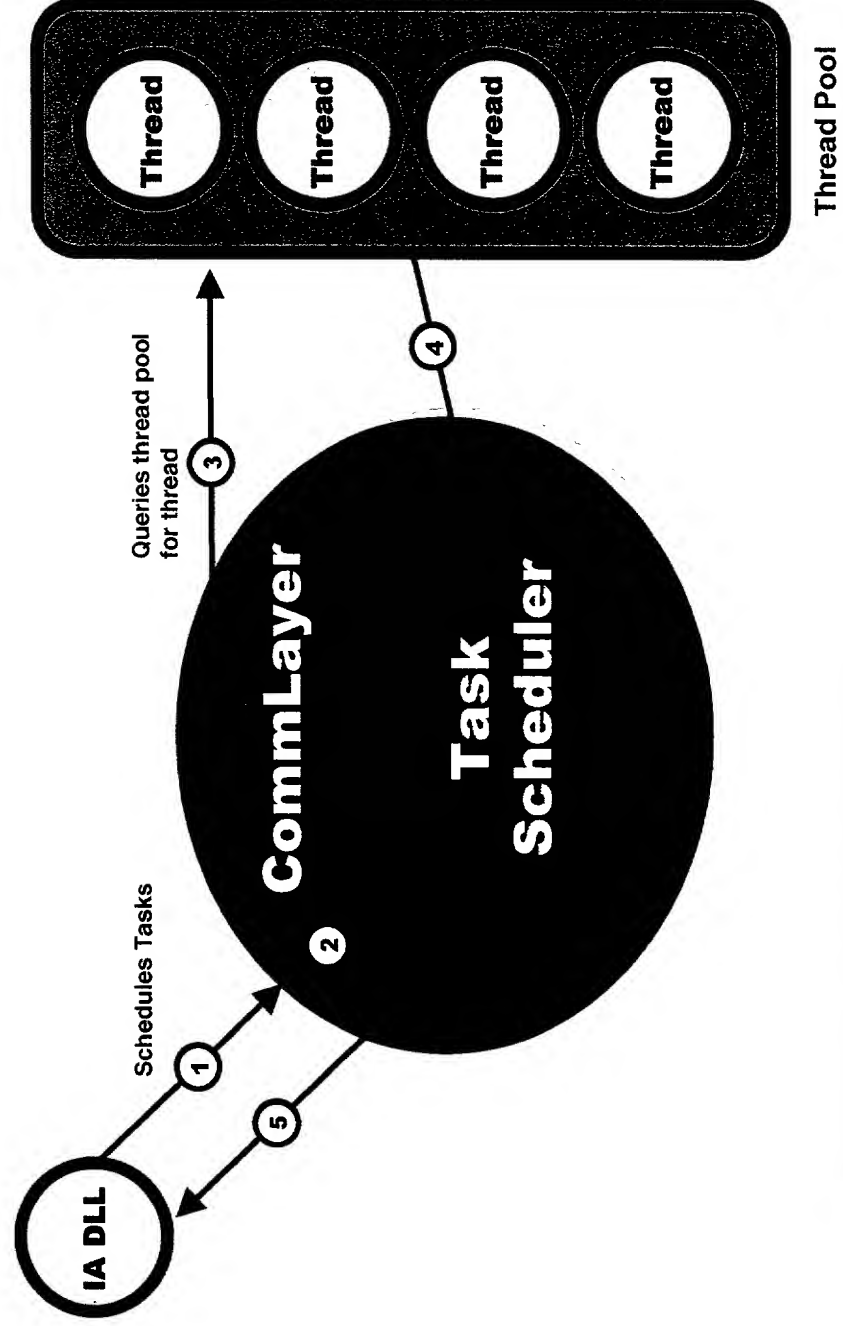
CommLayer Ping Functionality (3)



CommLayer Task Scheduling (1)

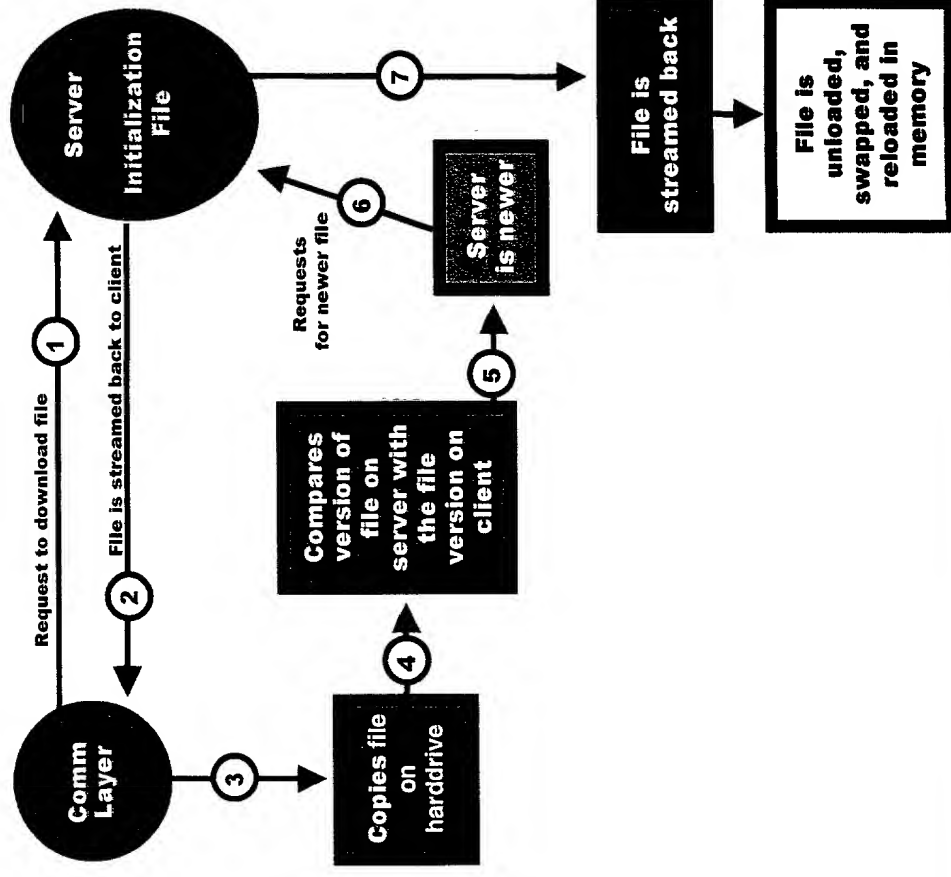
- **Manages the thread pool which avoids communication congestion and optimizes client performance**
- **Receives requests for tasks to be performed from the dependent DLLs (e.g., IA, Stats, etc.)**
- **Schedules and assigns tasks to the threads as threads become available in the thread pool**
- **“Do it now requests” - similar to task scheduling but provides threads immediately for tasks that cannot wait to be scheduled**
- **Governs overall number of threads that are made available depending upon the type of Internet connection**

CommLayer Task Scheduling (2)



CommLayer Hot Swapping

- Queries server for updated application components
- Downloads components as needed
- Registered DLLs are unloaded, swapped, and reloaded by the CommLayer and application files (e.g.: CommLayer and the xSides executable) are queued for swapping on system startup



CommLayer Dynamic Configurability

Application variables are stored on the server and cached locally – this allows tuning for all clients.

Examples of these variables are:

- Ping interval (time interval between two pings)
- Connectivity time interval
- Ping URL
- Upload URL

CommLayer Extensibility

- CommLayer initialization file is downloaded from the server periodically.
- This file holds the information for loading/unloading of DLLs and executables.
- It also has server version of these DLLs.
- CommLayer compares this version with the client side version, and if the server has newer version it downloads it and hot swaps it with the older version of the DLL.
- By using hot swapping and dynamic configurability, xSides can introduce a new DLL with new functionality, that conforms to the rules of the CommLayer.

ConnLayer Misc. Features

- Provides functions to upload and download files from the AllSides server (this functionality is used extensively by Marketplace DLL).
- This functionality can be extended to talk with any other server.

Instant Alert Overview

Instant Alerts provide a means for partners to send priority messages to their xSides users. Content is delivered to the client in standard HTML and displayed in a browser window. Users are identified by Global Unique Identifiers (GUID).

- Client/server architecture
- Markup driven using xSides communication layer
- Instant Alert content is HTML
- Server-side partner IP authentication
- API available to automate message generation

Instant Alert Features

Instant Alerts may be sent to single users or all users of an xSides side. Alerts may also be sent using templates, similar to a form letter, with or without replaceable parameters.

- Send Instant Alerts to a single user.
- Send Instant Alerts to all users of a side.
- Send templated Instant Alerts.
- Partner support for retrieving xSides user GUID.

Instant Alert Client Responsibility

- Schedules task thread with communication layer.
- Registers/unregisters for callback messages.
- Monitors for presence of URL in initialization file.
- Writes URL to initialization file.
- Responds to communications layer requests.

Instant Alert Data Flow

Below is a diagram of the sequence of requests and responses involved in retrieving an Instant Alert.

1 IA REQUEST

2 RESPONSE URL

3 URL REQUEST

4 INSTANT ALERT HTML

5 CONFIRMATION



Instant Alert Data Flow

Sequence of client requests and server responses:

- 1 Ping requests are made to the server
- 2 Response contains xSides markup which may contain a URL
- 3 If URL is present, it is written to an initialization file
- 4 xSides app monitors initialization file for URL
- 5 If URL is present, user is informed message waiting
- 6 User reads message by accessing Instant Alert URL
- 7 By closing Instant Alert, user accomplishes confirmation that the message was read

1 PING

2 RESPONSE URL

3 URL REQUEST

4 INSTANT ALERT HTML

5 CONFIRMATION



Instant Alert Templates

Instant alerts that are going to be used over and over with only minor changes to the content might be better suited to template style Instant Alerts. The template can be created once and used repeatedly by an xSides partner.

- Tools provided for creating and deleting templates.
- Tools provided for sending templated messages.
- Ability to create named attributes which get filled in when sent.

Instant Alert Partner Support

We provide a means for xSides partners to associate an xSides user's unique ID with the partner user's ID. The partner could create a button with a URL such as: <http://www.fabuloussite.com/scripts/Login.dll?Login&#pxid>

The “#pxid” is replaced with the user's GUID before it is exercised, thus providing the partner with the xSides user GUID. An xSides partner could then store this information with their own user ID. With this information in hand, the partner is able to send Instant Alerts to their xSides users.

- Partner user IDs can be associated with xSides user Ids.
- Macro replacement in URL provides access to xSides user GUID.

Instant Alert Client Class Diagram

The Instant Alert object derives from the communication layer, which manages server requests and delivers server responses.

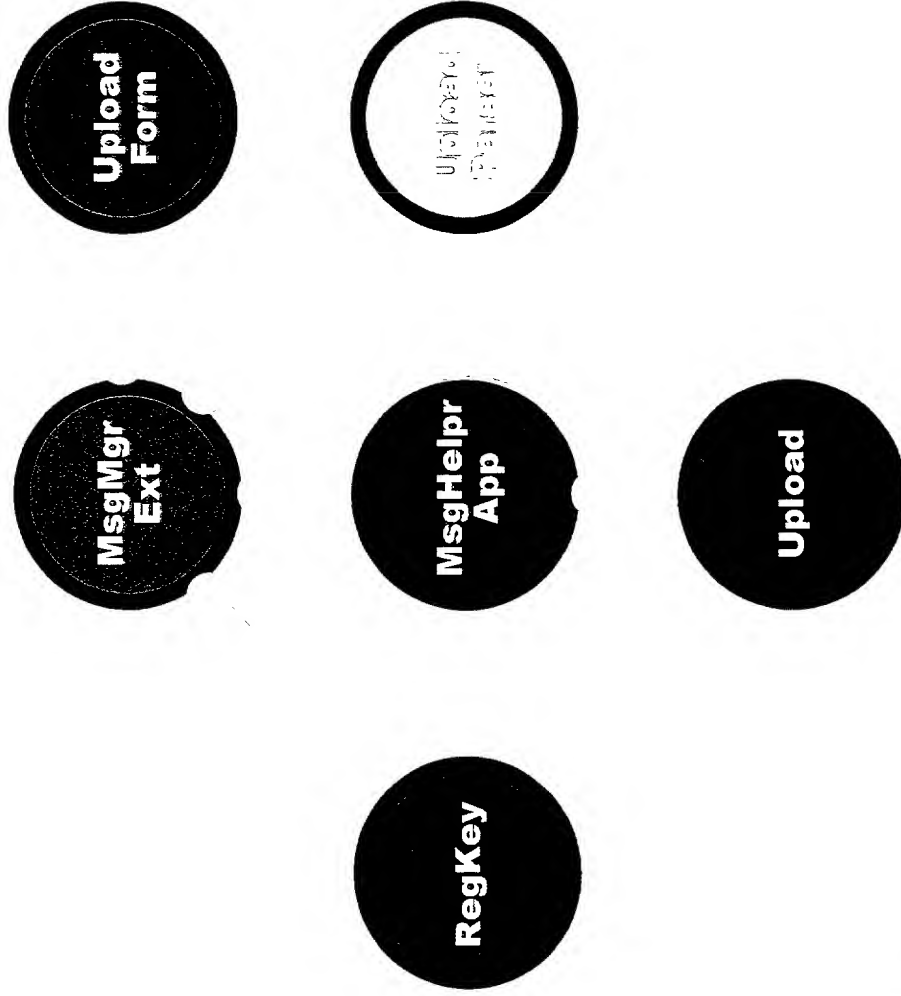
- Communication layer asks for requests from Instant Alert.
- Communication layer delivers responses via callback functions in Instant Alert.
- All client/server communication is through xSides markup.



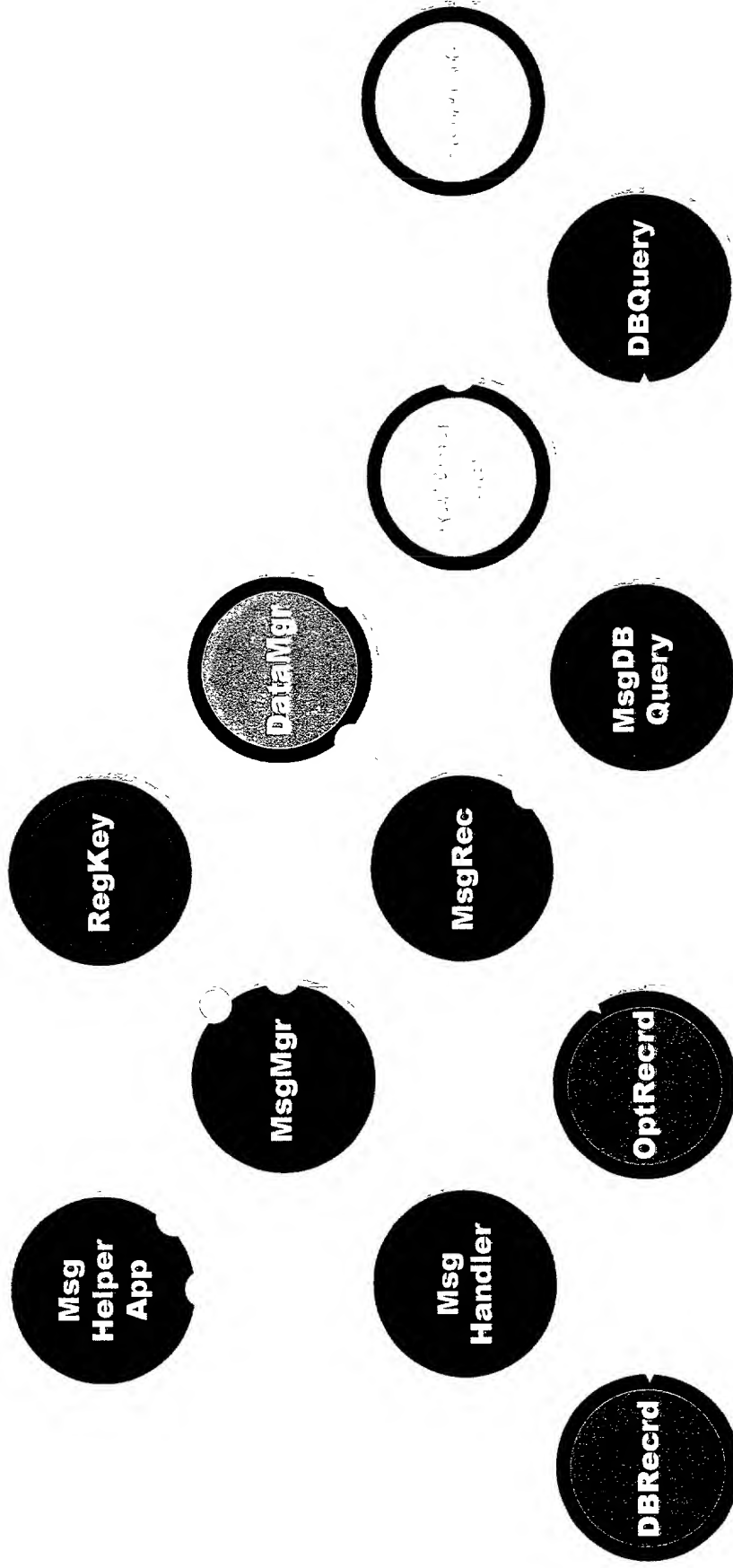
Instant Alert Server Responsibility

- **Delivers Instant Alert via HTTP request.**
- **Manages active user list.**
- **Provides IP security.**
- **Message delivery to and from database.**

Instant Alert Server Class Diagram A



Instant Alert Server Class Diagram B



Instant Alert Template Tool Overview

Instant Alert Templates can be created, modified and sent, all from one location.

Templates are a way to reuse a generic message for more than one user, and more than one time, like a form letter.

- Create templates in HTML
- Save and send from the same tool
- Easy to use, Web-based interface

The screenshot shows a web-based interface for the 'xSides instant alert! tool'. It features a table of available templates with columns for the template name and a count. Below the table is a text area for the template description. At the bottom right are buttons for 'Add New', 'Send', 'Delete', and 'Return to Developers Network'.

AVAILABLE TEMPLATES	
<input type="checkbox"/> Best	0
<input type="checkbox"/> Cort	0
<input type="checkbox"/> Crits's	0
<input type="checkbox"/> dup	1
<input type="checkbox"/> Disinfo	0
<input type="checkbox"/> dup3	0
<input type="checkbox"/> duplicate	0
<input type="checkbox"/> Duptest	4

TEMPLATE DESCRIPTION

Instant Alert Tool Template Selection Page

Lists all the templates for your company, as well as outstanding alerts for each template.

- Outstanding Alerts can be looked up quickly and easily
- Partner-defined descriptions remind you of a template's content
- Server-side partner authentication

xSides™ instant alert! tool

AVAILABLE TEMPLATES	OUTSTANDING ALERTS
<input type="checkbox"/> Best	0
<input type="checkbox"/> Cort	0
<input type="checkbox"/> Cris's	0
<input type="checkbox"/> dup	1
<input type="checkbox"/> Disinfo	0
<input type="checkbox"/> dup3	0
<input type="checkbox"/> duplicate	0
<input type="checkbox"/> Duptest	4
<input type="checkbox"/> ...	1

TEMPLATE DESCRIPTION

Add New

Send

Delete

Return to Developers Network

Instant Alert Tool Add New

Create a new template quickly and easily, using HTML. Templates and their descriptions show up immediately on the selection page, once they have been saved. Send as soon as you're finished.

- Create templates in HTML
- Dynamically replaceable fields
- 2,000 character template size
- Save in our secure database with a single click

The screenshot shows a web-based form titled "xSides™ instant alert! tool" with a sub-header "ADD NEW TEMPLATE". The form contains two text input fields: "TEMPLATE NAME" and "TEMPLATE DESCRIPTION". Below these is a large text area labeled "CUT & PASTE CODE:" containing HTML code for an alert. At the bottom right are three buttons: "Save", "Preview", and "Cancel".

TEMPLATE NAME

TEMPLATE DESCRIPTION

CUT & PASTE CODE:

```
<HTML><BODY>
<P> H1 <ARKTEKTNAME=FNAM>
Send me<ARKTEKTAMOUNT=NUMBER>
THIS IS a chain letter.
<BODY></HTML>
```

Save Preview Cancel

Instant Alert Tool Send

Intuitive Web-based interface lets you replace dynamic fields easily, and preview what your user will see. Set the expiration date and send to either an individual user or all users of a certain side.

- Simple replacement of dynamic fields by entering information into form
- Preview results identical to user experience
- Send to either individual user or all users of a single side

The screenshot shows the 'SEND TEMPLATE' interface of the xSides instant alert! tool. It is divided into several sections: 'REPLACEABLE FIELDS' with a 'Show Preview' button; 'SENDING OPTIONS' with a 'SINGLE USER' section containing a 'Globally Unique ID (GUID)' field and a 'BROADCAST' section with 'SIDE' and '#USERS' dropdown menus (currently showing 'Web/Search' and '136'); and a 'URL' section with 'Display URL' and 'Exp. Date' (set to '2000' and 'Feb') dropdowns, and 'Exp. Time: (h/m)' (set to '12' and '00') dropdowns. 'Send' and 'Cancel' buttons are at the bottom right.

Stats Overview

The Stats DLL is an object that resides on an end user's computer (the client) as a part of the xSides application.

When the xSides application is running on an end user's machine:

- Stats records the user's xSides activities.
- Recorded activity is sent to an xSides server and logged.
- Logged activity records can then be used for accounting purposes.

Stats Recorded Activities (1)

Examples of recorded activities are “clicks” and “impressions.”

- Click: a mouse click on a side or portal.
- Impression: the amount of time a given area of xSides is displayed to the client.
- Our Impression is a true impression. It is the length of time content is shown, i.e., true screen time.

e.g., if a user rotates the bar to Company A's side, then an impression will be the length of time this specific side is displayed. If the client clicks on this side, then that click is recorded.

Stats Recorded Activities (2)

The xSides executable tells the Stats DLL:

- What activity to record, (e.g., when the application displays an image on xSides, it tells the Stats DLL to start recording the impression time).
- When to record the activity (e.g., when the executable traps a mouse click, it then tells the Stats DLL to record the click).
- Where the activity happened: portal, or side.
- Which company's content the activity happened in.

Stats How It Works

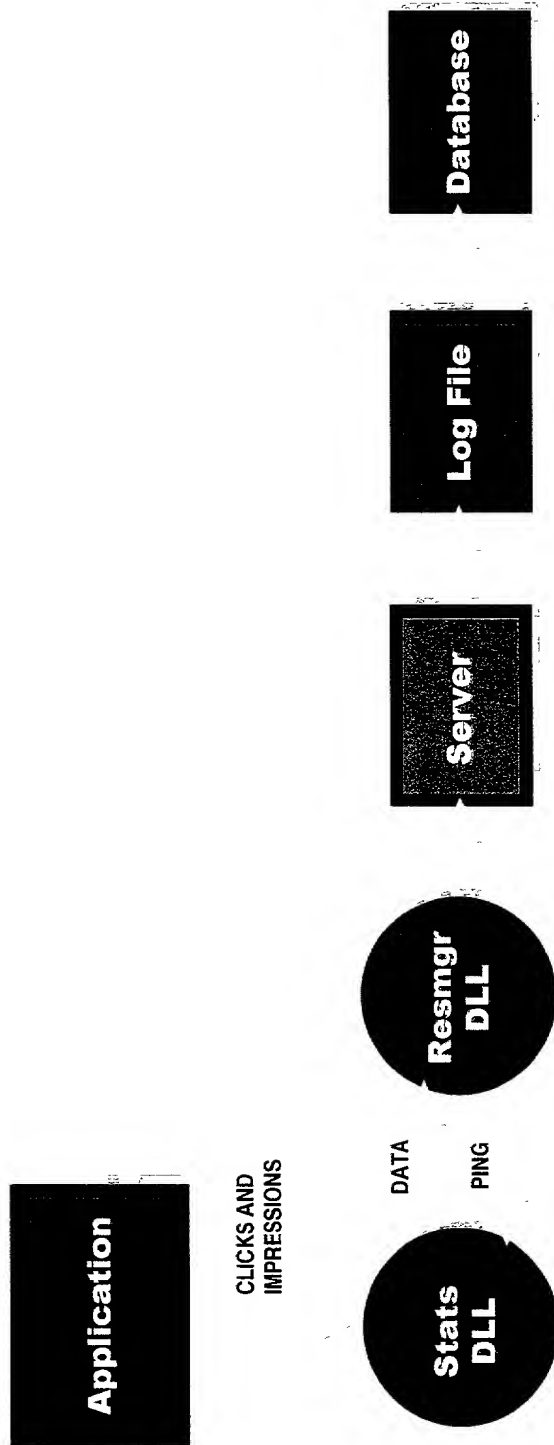
Application tells the Stats DLL to record an impression or click and passes the application name, the identifier of the content, its version, and the type of activity for recording to the Stats DLL.

1. A user clicks on a button in Company A's version 2.2 side.
2. The names –“side, Company A, 2.2, click” are recorded with the Stats component.
3. The number of clicks for “side, Company A, 2.2” is incremented by one.

Stats Information to the Server (1)

1. Resmgr DLL, another xSides DLL that resides on the client machine, pings Stats every minute at a scheduled interval.
2. Stats sends a markup string to the Resmgr DLL containing its data for the previous time period.
3. Resmgr DLL then uploads this to the server and the server logs this markup string into a log file for that day.
4. The Logger DLL knows how to parse these strings and enters the data into a database.
5. Invoices then can be generated from this database.

Stats Information to the Server (2)



CLICKS AND
IMPRESSIONS

Stats Validity of Impression Time

Impression time for an xSides side begins when it is first displayed to the user, and ends when another display replaces it.

1. Timeout value is set when Stats is first loaded.
2. When Stats first gets pinged by Resmgr, it receives a timeout value from the server.
3. For each successive ping, idletime value is sent to Stats.
4. Idletime value is compared to the timeout value.
5. When idletime value is greater than timeout value, impression time is cut off.

Other situations are handled similarly, such as minimum impression time.

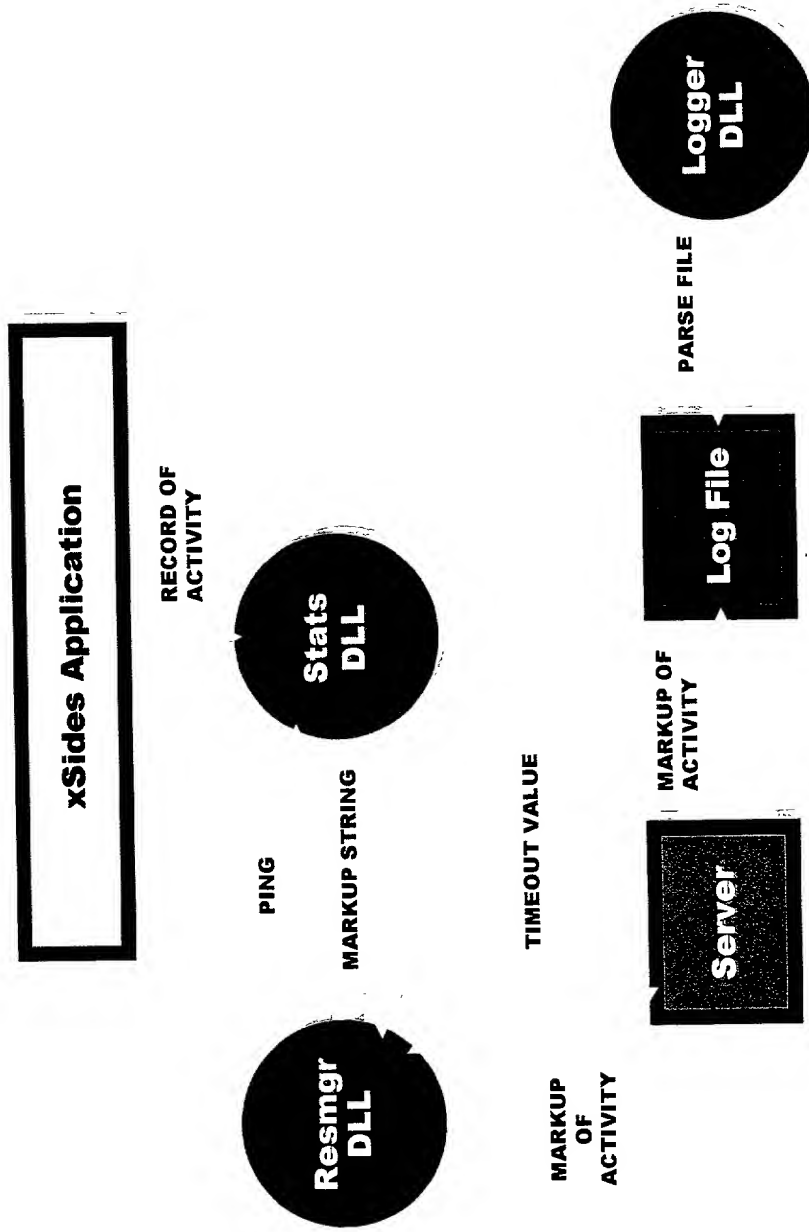
Stats Record of Activity

The data that is sent to the server to be logged is a string of characters that is in XML format:

```
<STATS><PORTAL><PIM Click=6 Impression=57 DVD=1.2></PIM><COMPANY A  
DVD=2.2></COMPANY A><COMPANY BDVD=2.3></COMPANY B></PORTAL><XSIDES><PIM  
DVD=4.0></PIM><COMPANY A Click=12 Impression=16 DVD=1.2></COMPANY  
A><COMPANY B Impression=34 DVD=1.2></COMPANY B></XSIDES></STATS>
```

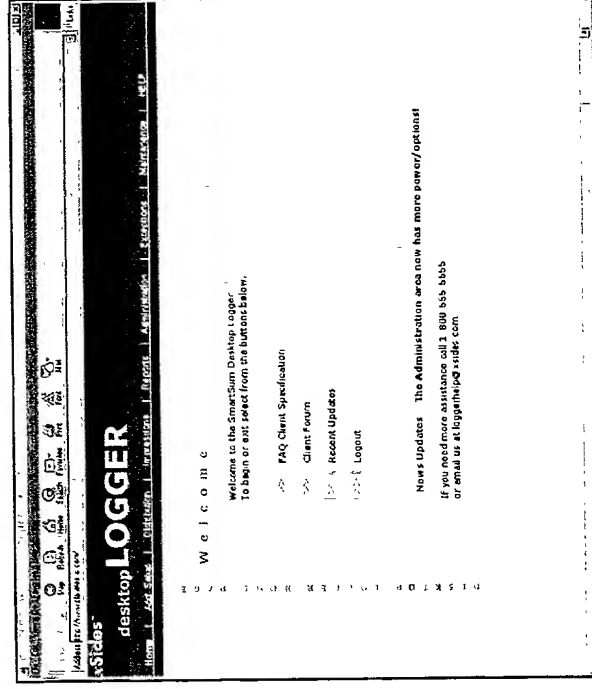
- String is downloaded to the server and logged.
- Logger processes the data for reporting and billing.

Stats Object Diagram



Desktop Logger Overview

- Desktop Logger calculates, stores, and retrieves activity-based statistics for a Web site or executable (xSides).
- Desktop Logger is used as an accounting tool that is rule-based: e.g., Desktop Logger uses pricing information to generate invoices and reports.



Desktop Logger Tracking

Web site activities that Desktop Logger tracks:

- Clicks
- Users (Through an anonymous ID)
- Visits (Defined by time intervals)
- Unique visits (The first visit only)
- Destination URLs
- Source location (xSides side)
- Version/distribution information

Desktop Logger How it Works (1)

Step 1

Two methods send markup to our xSides servers:

- Redirected links
- “Ping” from an application

What is markup?

Markup allows us to send detailed information to our servers that would not normally be sent.

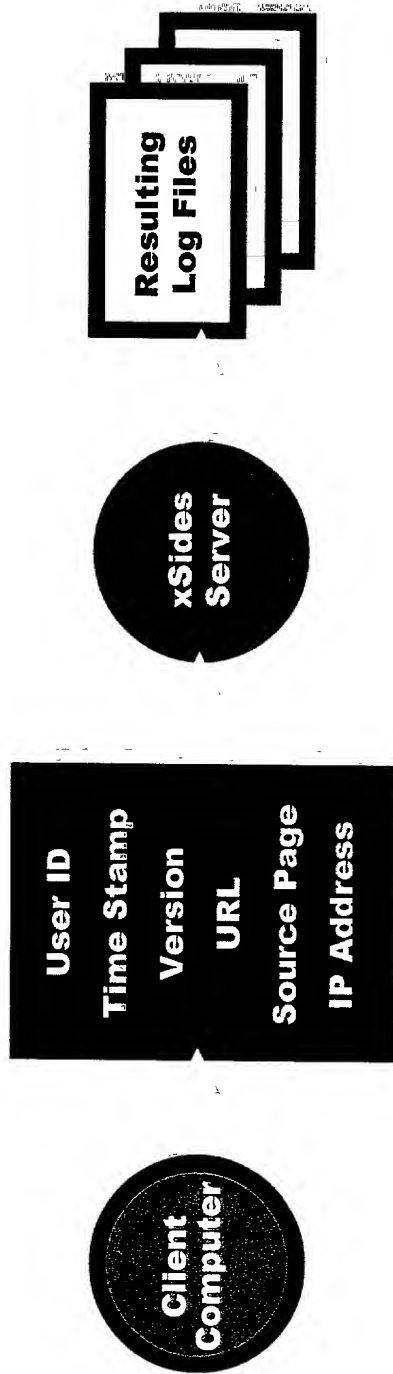
Sample Decrypted Markup:

```
<REQUEST%11GUID=0M555A6180-B6EC-11D3-BD0F-00902716AAB6%11SIGNON=1%09APPVER=PXGEN><PARAM%09AC=GET></PARAM><INI_VER%09AC=GETLIST></INI_VER><STATS%09AC=GET></STATS><MKT%09AC=SIGNON></MKT><MKT%11AC=GETSIDELEFT></MKT></REQUEST>
```

The markup goes into the natural storage mechanism of the Web server so it can't be altered.

Desktop Logger How it Works (2)

Possible Data Transferred Through Markup



Desktop Logger How It Works (3)

Step 2

- Desktop Logger parses the Internet logs.
- Compresses the pertinent data into a proprietary format.
- Inputs data into the back-end SQL database.

Step 3

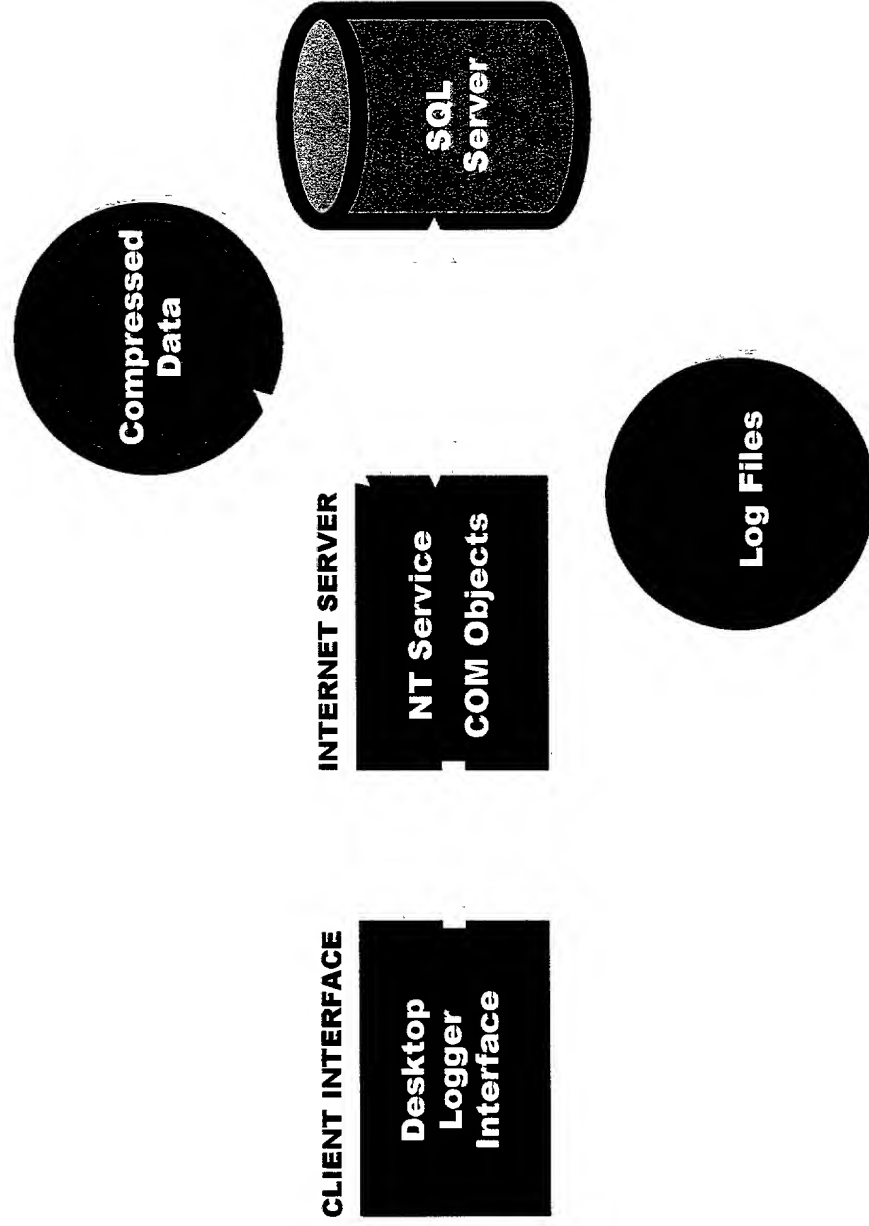
The Desktop Logger interface accesses the SQL database and compressed files in order to generate reports and invoices.

Desktop Logger 3-Tiered Architecture

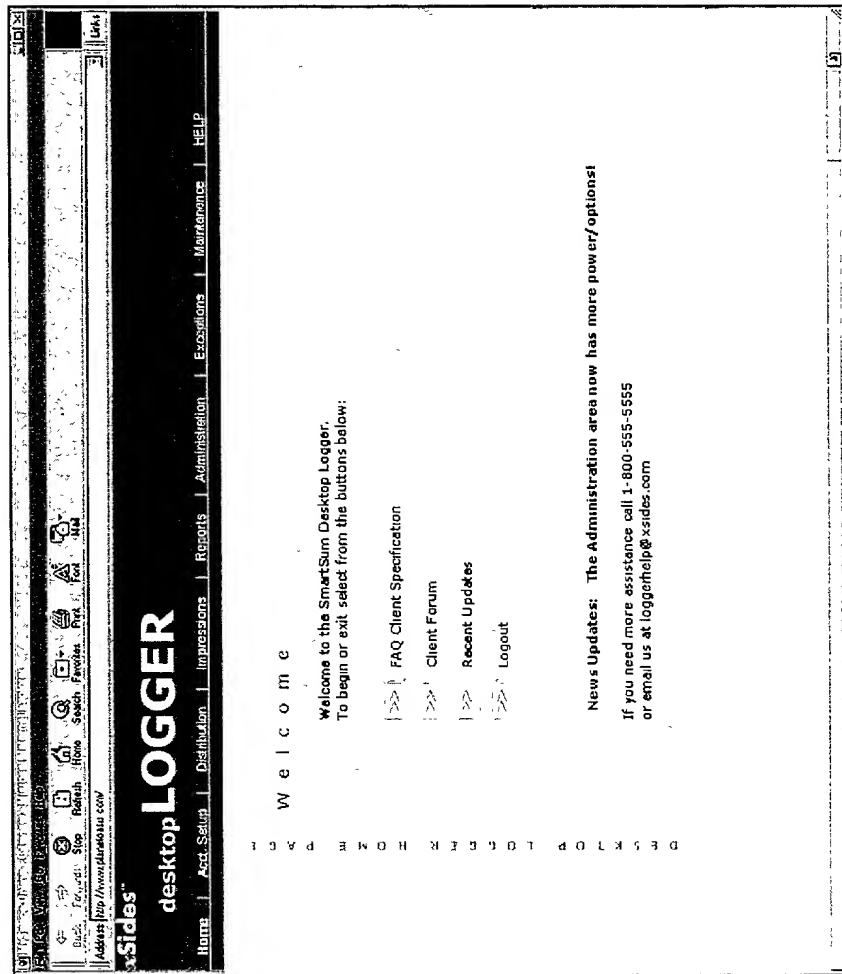
Desktop Logger is an application that consists of 3 main parts:

- The front-end is a Web-based client interface called Desktop Logger.
- Middle tier processes raw IIS server logs – the work horse of Desktop Logger.
- Backend consists of a large SQL database that stores for partner, distributor and statistical information.

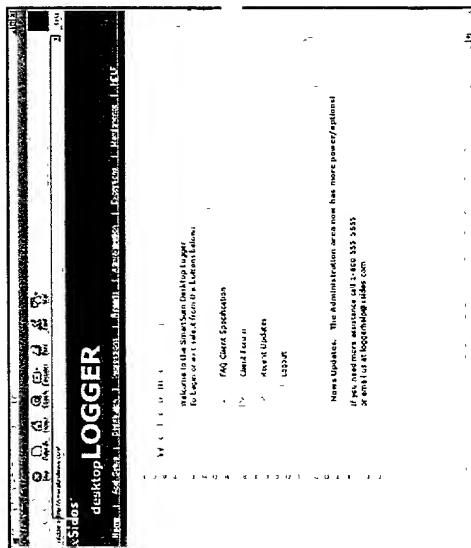
Desktop Logger 3-Tiered Architecture



Desktop Logger Front-End

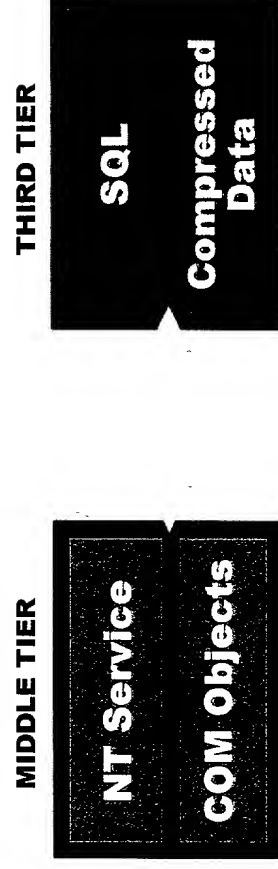


Desktop Logger Front-End



Desktop Logger Middle Tier

- Processes raw log files automatically from numerous servers.
- Compresses data into a proprietary format.
- Automatically inserts data into SQL database.
- Supports ActiveX objects that are accessible through a Web interface to generate reports and invoices.



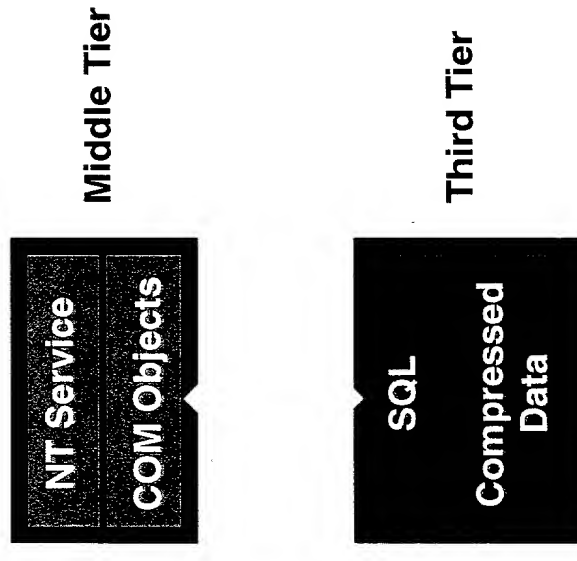
Desktop Logger Third Tier

SQL database contains:

- Tables
- Stored procedures
- Triggers

SQL database stores:

- Invoicing statistics
- Pricing rules
- Operator logins and security levels
- Company information
- Version, source, URL information
- Exceptions



[illegible]

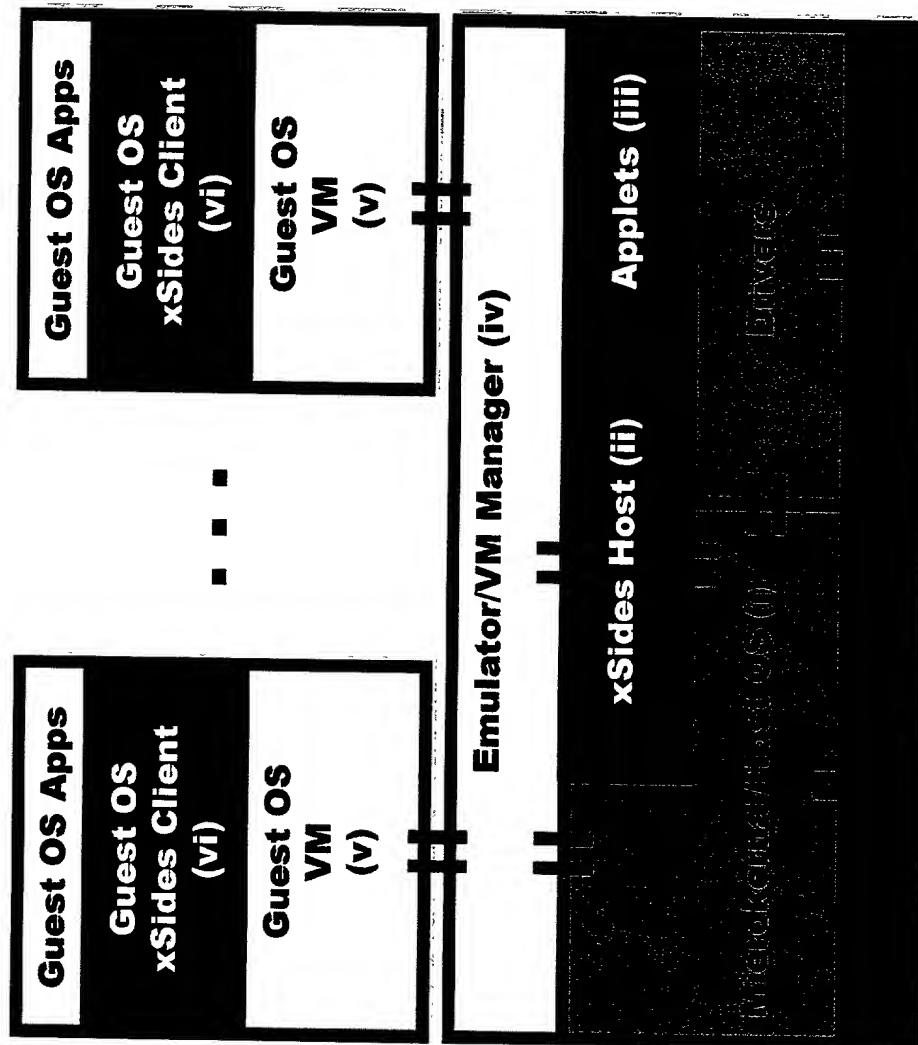
The Microkernel Project (1)

The Microkernel is a secondary operating system for separately supporting xSides enabled applets.

PROJECT DESCRIPTION:

- The Microkernel will enable the xSides interface to function even if the primary operating system for the desktop fails (i.e., xSides is "always on").
- xSides to port the Microkernel (including xSides drivers and applications, as applicable) to a set-top box environment.

The Microkernel Project (3)



The Applets Project

Each Applet to run with or without the Microkernel.

STREAMING AUDIO AND VIDEO PROJECT

PROJECT DESCRIPTION: xSides to develop software applets for enabling the transmission of two-way audio and video including, e.g., one or more of the following: VoIP technology, IP Streaming Video technology, Video Encoding technology and Video Conferencing technology.

TV TUNER PROJECT

PROJECT DESCRIPTION: xSides to develop a software applet for enabling television programming applications utilizing, e.g., Broadcom's proprietary 2D/3D graphics and Advanced Tuner Technology (EPG is an example).

The Applets Project

VoIP PROJECT

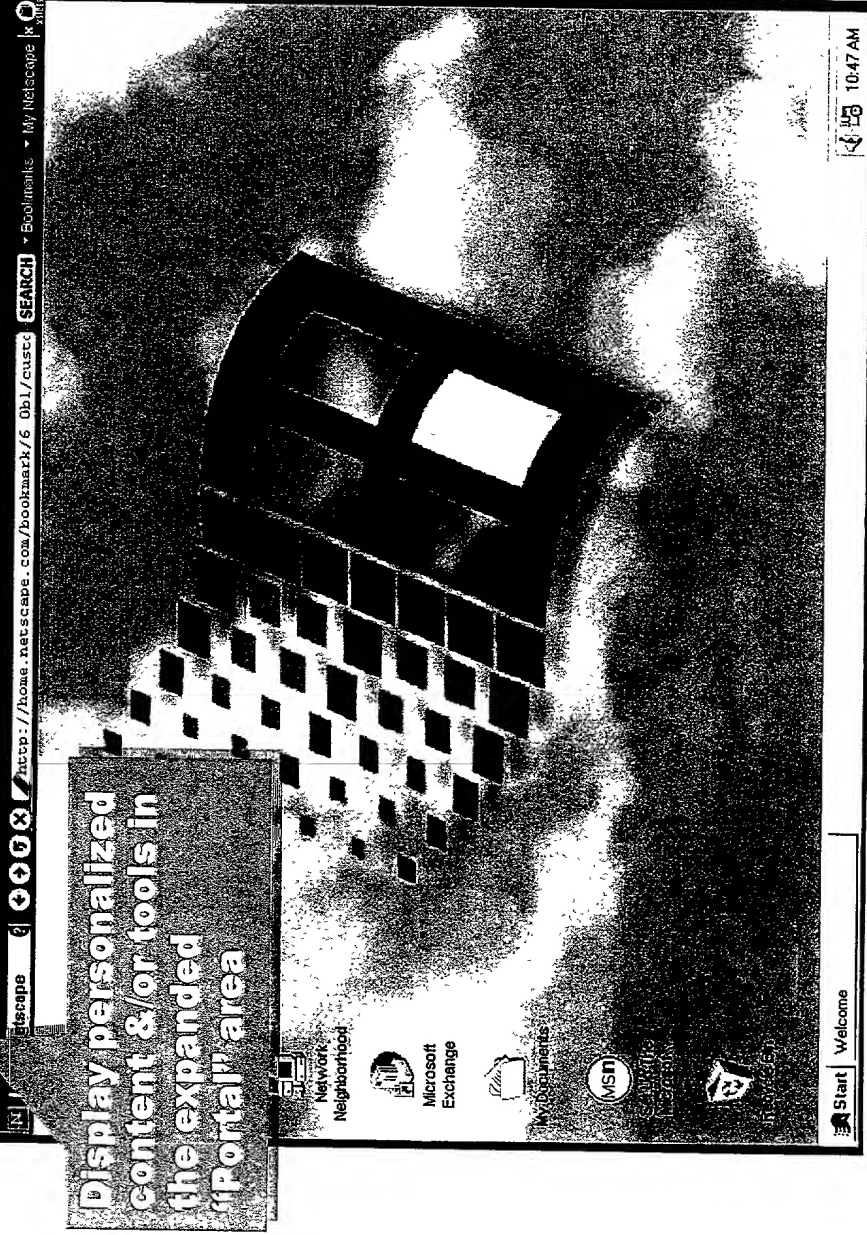
PROJECT DESCRIPTION: xSides to develop software applets for enabling a user interface and the transmission of voice over broadband networks using VoIP, VoDSL, and VoATM technology.

HDTV PROJECT

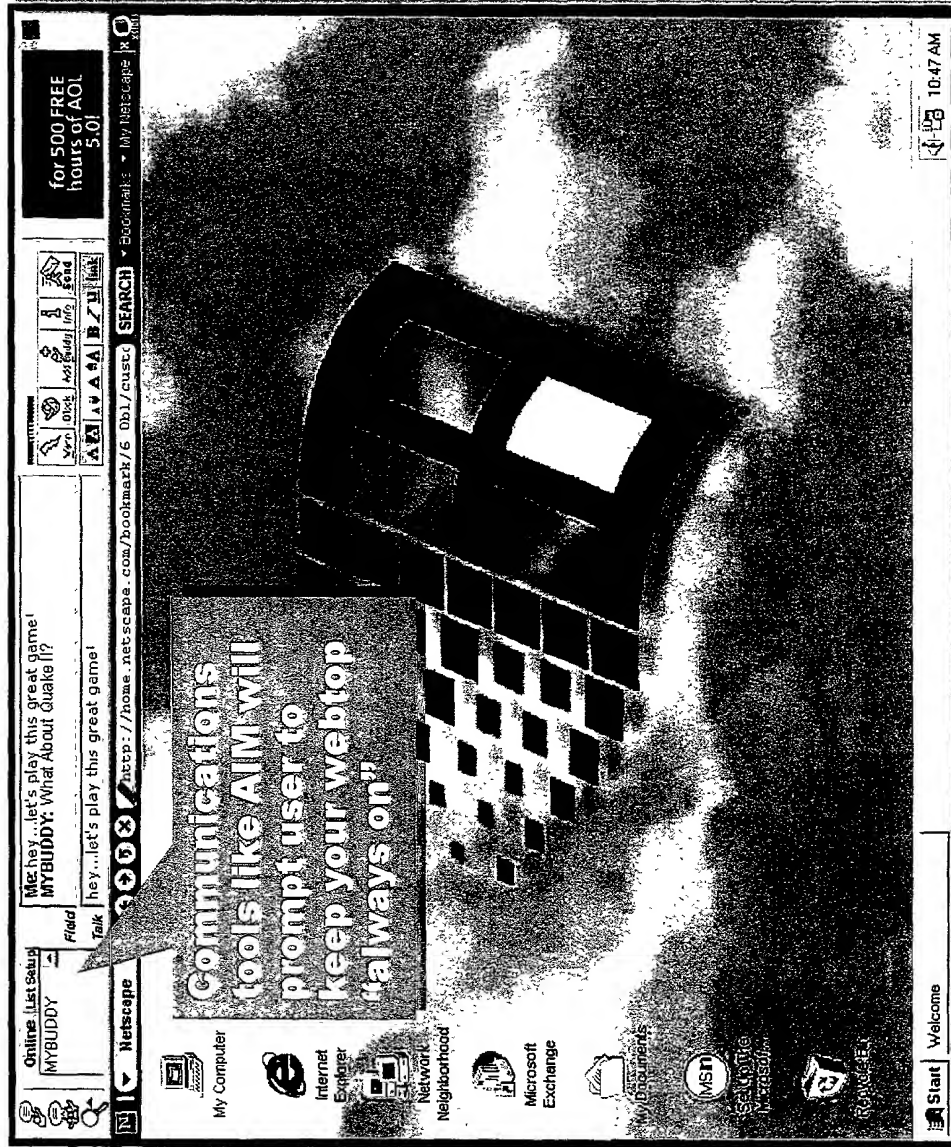
PROJECT DESCRIPTION: xSides to develop software applets for enabling the display of HDTV on PCs, set-tops, and IP appliances, including, e.g., one or more of the following: HDTV receiver, HDTV MPEG decoder, and 2D/3D graphics technology.

Product Design Examples

This is your space. Assume control of your own real desktop.
Place HTML, DHTML, Flash, C, C++, or use any other application you can imagine.
And have the same foothold as a OS with the flexibility of a thin client.



Product Design Examples



∞

Product Design Examples

My Computer
 Internet Explorer
 Network Neighborhood
 Microsoft Exchange

X-COMPANY
instant alert!
Your Order is Shipped!
 Item #008440
 Quantity 306 at \$59.95

Flashing Instant Alerts enable priority communication

Portal area can support your communications for employees, vendors and customers

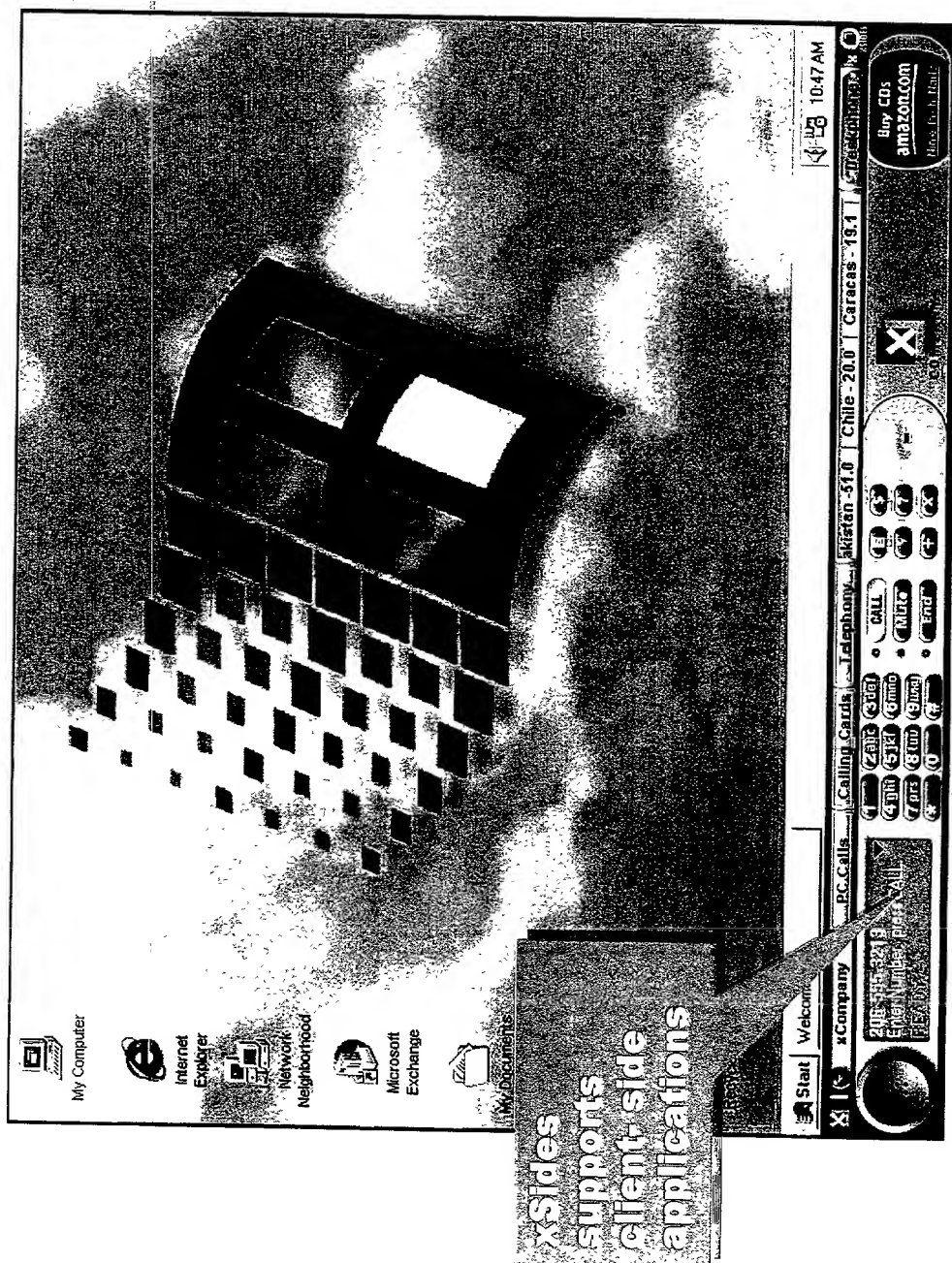
Product	In Stock	Backordered	In Stock	Inventory	Inventory
Product #000001	In Stock		\$348.98	Add	
Product #000002	In Stock		\$1,099.95	Add	
Product #000003	Backordered			Add	
Product #000004	In Stock		\$32.95	Add	
Product #000005	In Stock		\$14,998.00	Add	

TODAY'S STEAL
New and improved at an introductory \$1,099

ADVISOR

Variable	Mean	SD	Min	Max
Age	35.2	12.5	18	65
Gender	Male	10.5	0	20
Marital status	Married	15.2	0	20
Education	High school	12.5	0	20
Occupation	Unemployed	10.5	0	20
Income	Low	10.5	0	20
Health status	Good	15.2	0	20
Stress level	High	10.5	0	20
Life satisfaction	Low	10.5	0	20
Depression	High	10.5	0	20
Anxiety	High	10.5	0	20
Substance use	Low	10.5	0	20
Physical activity	Low	10.5	0	20
Social support	Low	10.5	0	20
Resilience	Low	10.5	0	20
Optimism	Low	10.5	0	20
Self-efficacy	Low	10.5	0	20
Problem-solving	Low	10.5	0	20
Emotional regulation	Low	10.5	0	20
Interpersonal skills	Low	10.5	0	20
Life goals	Low	10.5	0	20
Meaning in life	Low	10.5	0	20
Existential well-being	Low	10.5	0	20
Psychological well-being	Low	10.5	0	20
Overall well-being	Low	10.5	0	20

Product Design Examples



Product Design Examples

